Universidad Zaragoza

Proyecto de fin de Carrera
Ingeniería en Informática
Curso 2010/2011

Personajes con Razonamiento Basado en Casos para videojuegos en primera persona

Javier Olmos Lanceta

Director: Manuel González Bedia
Co-Director: Francisco Serón Arbeloa

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

Septiembre de 2011

Personajes con Razonamiento Basado en Casos para videojuegos en primera persona

RESUMEN

Actualmente los videojuegos han llegado a un punto de realismo gráfico difícil de superar y aun así cualquier jugador experto reconoce que el realismo en la interacción es deficitario. El problema de fondo, al que se ha empezado a dedicar tiempo y dinero en los últimos años no es gráfico, sino que se centra en cómo mejorar aspectos de comportamiento, como la inteligencia artificial, para dotar de mayor credibilidad a los personajes virtuales. Siguiendo este enfoque, ha aparecido recientemente un nuevo interés por aplicar técnicas de modelado cognitivo que buscan que los personajes virtuales muestren mayor semejanza con el comportamiento humano.

El objetivo de este proyecto es usar técnicas cognitivas para crear un personaje que se comporte, en su conducta, de la manera más humana posible. La técnica seleccionada para este fin ha sido el uso de un sistema de razonamiento basado en casos para implementar la hipótesis de los "marcadores somáticos" de Antonio Damasio [13]. El modelo de Damasio es la teoría más aceptada sobre el modo en que los humanos toman decisiones, y cómo se ven afectadas por las emociones.

El funcionamiento del modelo de decisión implementado es como sigue: El personaje tiene una memoria con experiencias pasadas y un valor que le indica si fue buena o mala dicha experiencia. En un momento de la partida busca en su memoria cómo actuar ante la situación que encuentra. En su memoria se hará una criba previa de casos asignados como positivos por su experiencia y el personaje elegirá el más parecido a la situación actual.

Lo primero que exige este proyecto es estudiar qué herramientas tenemos disponibles para llevarlo a cabo. Se estudiarán a fondo herramientas como Pogamut y SQLite. Pogamut son un conjunto de librerías en java para el manejo de personajes del Unreal Tournament 2004. SQLite es un gestor de bases de datos, muy simple y completo con gran rápidez de ejecución perfecto para juegos en tiempo real.

Una vez concluido ese trabajo previo, se analizará cómo queremos que se comporte el personaje. Como el entorno de trabajo es muy amplio hay que simplificarlo. Se deben elegir los parámetros más importantes que el personaje debe tener en cuenta para elegir el comportamiento futuro, como el entorno de un juego da demasiada información hay que seleccionar la más adecuada. Además se crean los comportamientos que el personaje podrá ejecutar, intentando darle diversidad de acción, para poder controlar las diversas situaciones que se encuentre en la partida. Los comportamientos serán árboles de binarios de tamaño variable entre cuatro y siete niveles dependiendo de las acciones que lo conformen.

Los resultados finales obtenidos demuestran que las técnicas implementadas son útiles e interesantes para seguir avanzando en trabajos futuros.

Agradecimientos

Durante los años que ha durado la carrera, incluyendo este último hay mucha gente a la que tengo que estar muy agradecido por haberme apoyado en los buenos y malos momento, ya sea con pequeños o grandes gestos.

A mi familia, especialmente a mis padres, Lourdes y Jesús, y sus respectivas parejas, Josetxo y Diana, por aguantarme tantos años escuchando cosas que seguramente no entendían, aunque como siempre me ha dicho mi madre le gustaba escucharme porque veía que me divertía con lo que hacía. También mis hermanas pequeñas, Ana e Izaskun, me han ayudado mucho aunque no lo sepan, porque siempre conseguían que desconectara al sacar mi lado gracioso o picajoso con ellas. Aunque siempre se dice que la familia no se elije, yo no puedo quejarme de la que me ha tocado.

A mis amigos (tanto de infancia, como del cps) que, como siempre, han estado ahí durante toda la carrera y en especial este último año, interesándose por mis avances y haciendo preguntas sobre el proyecto que yo no me había hecho y que me han ayudado a avanzar. Además también les tengo que agradecer, y mucho, el ser esas válvulas de escape necesarias para desconectar de todo y disfrutar de una vida social por lo menos aceptable, tan importante es terminar un trabajo con la mejor nota posible como disfrutar de una buena fiesta con los amigos.

A todos los habitantes del cps que han compartido tiempo, sufrimientos y alegrías conmigo. Y a Sergio, Carlos y Ángel por habernos ayudado mutuamente en lo posible a controlar todo el entorno del juego.

A mis tutores Paco y Manolo por ayudarme en todo lo posible, y en especial por haberme permitido juntar en un mismo proyecto dos de los temas que más me atraen, videojuegos y cerebro. Su ayuda en todo este tiempo ha sido fundamental para mí, tanto en los buenos momentos, dejando claro que no todo estaba tan bien, como en los malos, mostrando que todo tiene solución aunque no parezca cercana.

Tal como dijo la escritora Gladys Bronwyn Stern, “La gratitud en silencio no sirve a nadie”, así que yo solo puedo decir,

GRACIAS.

Índice general

1. Introducción **1**

1.1. Objetivos 1

1.2. Contenido de la memoria 2

1.3. Contexto 2

1.3.1. Emociones y videojuegos 3

2. Cuestiones exploradas o trabajo previo **7**

2.1. Pogamut 7

2.1.1. Cómo crear un bot 8

2.2. SQLite 13

2.3. Técnicas y herramientas desechadas 13

3. Descripción del modelo **15**

3.1. Condiciones de la partida 15

3.2. Variables de entorno 16

3.3. SomaticMarkerBot 17

3.3.1. Comportamientos 19

4. Diseño de Experimentos y Resultados **23**

4.1. Parámetros de los experimentos 23

4.1.1. Ventana 23

4.1.2. Métricas de selección de caso 23

4.1.3. Métricas “premio/castigo” 25

4.2. Experimentos 25

4.2.1. Experimento 1 25

4.2.2. Experimento 2 27

4.2.3. Experimento 3 29

4.2.4. Experimento 4 31

4.2.5. Experimento 5 32

4.3. Ideas generales 34

5. Conclusiones y valoración personal **35**

5.1. Conclusiones . . . 35

5.2. Trabajo o líneas futuras . . . 39

5.3. Valoración personal . . . 40

A. Planificación **47**

B. Botprize **55**

B.1. Reglas de competición . . . 55

B.1.1. The task . . . 55

B.1.2. To enter . . . 55

B.1.3. Testing protocol . . . 56

B.2. Resultados botprize 2010 . . . 56

B.3. Resultados de los jueces . . . 57

C. Teoría de Damasio **59**

C.1. Modelo emocional . . . 59

C.1.1. Emociones primarias . . . 60

C.1.2. Emociones secundarias . . . 60

C.1.3. Estados corporales "como si" . . . 62

C.2. La fría cognición no es suficiente . . . 63

C.2.1. Hipótesis del marcado somático . . . 64

D. CBR: Técnicas de Razonamiento basado en Casos **67**

D.1. Características generales de las 4 fases . . . 67

D.1.1. Recuperación de Casos . . . 67

D.1.2. Reutilización de Casos . . . 68

D.1.3. Revisión de Casos . . . 69

D.1.4. Retención de Casos . . . 69

D.2. Variaciones del CBR típico . . . 70

D.3. CBR y otras técnicas . . . 71

E. Instalación del entorno **73**

E.1. Prerrequisitos . . . 73

E.2. Proceso de instalación . . . 73

E.3. Ejecución de un bot en UT2004 . . . 74

E.3.1. Configuración del servidor . . . 74

E.3.2. Conexión del bot al servidor . . . 75

F. Lista de comandos y mensajes **79**
F.1. Lista de comandos . . . 79
F.2. Lista de mensajes . . . 80

G. Listeners **83**

H. Variables de entorno **89**

I. Funciones de la base de casos **101**

J. Acciones simples **103**
J.1. Caminar (Walk) . . . 103
J.2. Recoger objetos . . . 104
J.3. Huir (RUN AWAY FROM PLAYER) . . . 105
J.4. Atacar . . . 108
J.4.1. Pasos previos . . . 109
J.4.2. Buscar enemigo . . . 109
J.4.3. Elegir arma . . . 111
J.4.4. Calcular distancia . . . 113
J.4.5. Hay rival y tengo arma . . . 113
J.4.6. Hay rival, pero lejos . . . 114

K. Árboles de comportamiento **115**

L. Experimentos **129**
L.1. Experimento 1 . . . 129
L.1.1. Caso 1 . . . 129
L.1.2. Caso 2 . . . 134
L.1.3. Caso 3 . . . 138
L.1.4. Caso 4 . . . 142
L.2. Experimento 2 . . . 146
L.2.1. Caso 1 . . . 146
L.2.2. Caso 2 . . . 151
L.2.3. Caso 3 . . . 155
L.2.4. Caso 4 . . . 159
L.3. Experimento 3 . . . 163
L.3.1. Caso 1 . . . 163
L.3.2. Caso 2 . . . 168

L.4. Experimento 4 172
L.4.1. Caso 1 172
L.4.2. Caso 2 177
L.5. Experimento 5 181
L.5.1. Caso 1 181

Índice de figuras

1.1. Explicación gráfica del test de Turing 3
1.2. Ciclo de vida de un sistema CBR. 5

2.1. Arquitectura de Pogamut. 7
2.2. Arquitectura a alto nivel de GaviaLib 8

3.1. CBR clásico y CBR UT2004. 18
3.2. Árbol binario 19
3.3. Árbol de comportamiento. 21

4.1. Victorias del bot 26
4.2. Comportamientos de ataque y defensa 26
4.3. Recoger objetos y Tipos de ataque 27
4.4. Inicio base de casos 27
4.5. Inicio base de casos 28
4.6. Victorias del bot 28
4.7. Comportamientos de ataque y defensa 29
4.8. Recoger objetos y Tipos de ataque 29
4.9. Victorias del bot 30
4.10. Comportamientos de ataque y defensa 30
4.11. Recoger objetos y Tipos de ataque 30
4.12. Inicio base de casos 31
4.13. Victorias del bot 31
4.14. Comportamientos de ataque y defensa 32
4.15. Recoger objetos y Tipos de ataque 32
4.16. Inicio base de casos 32
4.17. Victorias del bot 33
4.18. Comportamientos de ataque y defensa 33
4.19. Inicio base de casos 33

4.20. Recoger objetos y Tipos de ataque 34

A.1. Separación de horas por tipo de trabajo 52
A.2. Separación de las horas de Implementación 52
A.3. Separación de horas por tipo de trabajo (detallado) 53

C.1. Representación del cerebro. 59
C.2. Proceso de la emoción primaria. 60
C.3. Proceso de la emoción secundaria. 62
C.4. Caminos de procesamiento de información. 63

D.1. Ciclo de vida de un sistema CBR. 67
D.2. Posibles tecnologías usadas en la metodología CBR. 71

E.1. Pantalla de selección de juego. 74
E.2. Reglas de servidor. 75
E.3. Acceso a código fuente en Netbeans. 76
E.4. Añadir servidor del juego. 76
E.5. Configurar servidor del juego. 77
E.6. Entrar como espectador al juego. 77
E.7. Pantalla de debug dentro del juego. 78

K.1. Comportamiento de ataque: Ataque 1 y Objetos 1. 116
K.2. Comportamiento de ataque: Ataque 1 y Objetos 2. 117
K.3. Comportamiento de ataque: Ataque 2 y Objetos 1. 118
K.4. Comportamiento de ataque: Ataque 2 y Objetos 2. 118
K.5. Comportamiento de ataque: Ataque 3 y Objetos 1. 119
K.6. Comportamiento de ataque: Ataque 3 y Objetos 2. 119
K.7. Comportamiento de ataque: Ataque 4 y Objetos 1. 120
K.8. Comportamiento de ataque: Ataque 4 y Objetos 2. 120
K.9. Comportamiento defensivo: Ataque 1 y Objetos 1. 121
K.10.Comportamiento defensivo: Ataque 1 y Objetos 2. 122
K.11.Comportamiento defensivo: Ataque 2 y Objetos 1. 123
K.12.Comportamiento defensivo: Ataque 2 y Objetos 2. 123
K.13.Comportamiento defensivo: Ataque 3 y Objetos 1. 124
K.14.Comportamiento defensivo: Ataque 3 y Objetos 2. 125
K.15.Comportamiento defensivo: Ataque 4 y Objetos 1. 126
K.16.Comportamiento defensivo: Ataque 4 y Objetos 2. 127

Índice de tablas

2.1. Tipos de objetos en UT2004 11

3.1. Nivel de experiencia de los bot UT2004. 15
3.2. Mutadores del juego. 15
3.3. Parámetros globales de la partida. 16
3.4. Acciones simples. 20

A.1. Planificación. 51
A.2. Separación de horas por tipo de trabajo. 52

B.1. Humanidad de los bots. 56
B.2. Humanidad de los jugadores humanos. 57
B.3. Precisión de los bot al juzgar. 57
B.4. Precisión de los jueces al juzgar. 57

F.1. Lista de comandos de GameBots2004. 80
F.2. Lista de mensajes de GameBots2004. 81

H.1. Variables de entorno obtenidas del módulo Game. 90
H.2. Variables de entorno obtenidas del módulo AgentInfo. 94
H.3. Variables de entorno obtenidas del módulo Players. 94
H.4. Variables de entorno obtenidas del módulo Senses. 96
H.5. Variables de entorno obtenidas del módulo Items. 97
H.6. Variables de entorno obtenidas del módulo Weaponry. 99

J.1. Tipos de objetos que el juego permite recoger. 104
J.2. Funciones de recogida de objetos seleccionadas. 104

L.1. Tabla de victorias 130
L.2. Comportamientos generales 130
L.3. Comportamientos de ataque 130

L.4. Comportamientos defensivos 130
L.5. Formas de recoger objetos 131
L.6. Formas de atacar 131
L.7. Memoria del bot Exp:1 Cas:1 133
L.8. Tabla de victorias 134
L.9. Comportamientos generales 134
L.10. Comportamientos de ataque 134
L.11. Comportamientos defensivos 135
L.12. Formas de recoger objetos 135
L.13. Formas de atacar 135
L.14. Memoria del bot Exp:1 Cas:2 137
L.15. Tabla de victorias 138
L.16. Comportamientos generales 138
L.17. Comportamientos de ataque 138
L.18. Comportamientos defensivos 139
L.19. Formas de recoger objetos 139
L.20. Formas de atacar 139
L.21. Memoria del bot Exp:1 Cas:3 141
L.22. Tabla de victorias 142
L.23. Comportamientos generales 142
L.24. Comportamientos de ataque 142
L.25. Comportamientos defensivos 143
L.26. Formas de recoger objetos 143
L.27. Formas de atacar 143
L.28. Memoria del bot Exp:1 Cas:4 145
L.29. Tabla de victorias 146
L.30. Comportamientos generales 146
L.31. Comportamientos de ataque 147
L.32. Comportamientos defensivos 147
L.33. Formas de recoger objetos 147
L.34. Formas de atacar 147
L.35. Memoria del bot Exp:2 Cas:1 150
L.36. Tabla de victorias 151
L.37. Comportamientos generales 151
L.38. Comportamientos de ataque 151
L.39. Comportamientos defensivos 152

L.40. Formas de recoger objetos . . . 152
L.41. Formas de atacar . . . 152
L.42. Memoria del bot Exp:2 Cas:2 . . . 154
L.43. Tabla de victorias . . . 155
L.44. Comportamientos generales . . . 155
L.45. Comportamientos de ataque . . . 155
L.46. Comportamientos defensivos . . . 156
L.47. Formas de recoger objetos . . . 156
L.48. Formas de atacar . . . 156
L.49. Memoria del bot Exp:2 Cas:3 . . . 158
L.50. Tabla de victorias . . . 159
L.51. Comportamientos generales . . . 159
L.52. Comportamientos de ataque . . . 159
L.53. Comportamientos defensivos . . . 160
L.54. Formas de recoger objetos . . . 160
L.55. Formas de atacar . . . 160
L.56. Memoria del bot Exp:2 Cas:4 . . . 162
L.57. Tabla de victorias . . . 163
L.58. Comportamientos generales . . . 163
L.59. Comportamientos de ataque . . . 164
L.60. Comportamientos defensivos . . . 164
L.61. Formas de recoger objetos . . . 164
L.62. Formas de atacar . . . 164
L.63. Memoria del bot Exp:3 Cas:1 . . . 167
L.64. Tabla de victorias . . . 168
L.65. Comportamientos generales . . . 168
L.66. Comportamientos de ataque . . . 168
L.67. Comportamientos defensivos . . . 169
L.68. Formas de recoger objetos . . . 169
L.69. Formas de atacar . . . 169
L.70. Memoria del bot Exp:3 Cas:2 . . . 171
L.71. Tabla de victorias . . . 172
L.72. Comportamientos generales . . . 172
L.73. Comportamientos de ataque . . . 173
L.74. Comportamientos defensivos . . . 173
L.75. Formas de recoger objetos . . . 173

L.76. Formas de atacar 173
L.77. Memoria del bot Exp:4 Cas:1 176
L.78. Tabla de victorias 177
L.79. Comportamientos generales 177
L.80. Comportamientos de ataque 177
L.81. Comportamientos defensivos 178
L.82. Formas de recoger objetos 178
L.83. Formas de atacar 178
L.84. Memoria del bot Exp:4 Cas:2 180
L.85. Tabla de victorias 181
L.86. Comportamientos generales 181
L.87. Comportamientos de ataque 182
L.88. Comportamientos defensivos 182
L.89. Formas de recoger objetos 182
L.90. Formas de atacar 182
L.91. Memoria del bot Exp:5 Cas:1 185

Capítulo 1

Introducción

1.1. Objetivos

El objetivo fundamental del proyecto es la creación de un agente autónomo virtual (bot) cuyo comportamiento esté controlado por un sistema de Razonamiento Basado en casos (Case-based reasoning, CBR) que permita guardar experiencias pasadas con las que el agente organice su comportamiento futuro. Una vez implementado, el propósito es determinar las condiciones de una estrategia general para el comportamiento del bot, independiente de su situación concreta y del desarrollo de la partida.

Se ha elegido esta técnica para el modelado del agente por su sintonía con la teoría de los marcadores somáticos de Damasio. Nuestra experiencia previa, siguiendo las ideas de Damasio, quedaría reflejada en la base de casos mediante el valor de satisfacción "que posee cada uno" y nuestra capacidad deliberativa se modelaría mediante la métrica usada para elegir un caso en lugar de otro, después de haber discriminado los casos peor valorados. Como afirma Damasio: "en un proceso de decisión, inicialmente un valor emocional producido por experiencias pasadas restringe el número de opciones sobre las que elegir y posteriormente nuestra capacidad deliberativa elije la mejor opción de las restantes".

Entendido un sistema CBR como una metodología que hay que adaptar al contexto del trabajo, se propone que:

- El bot posee experiencias pasadas (casos) almacenadas en la base de casos. Cada caso es una pareja "situación del entorno - comportamiento".
- Según la situación actual del entorno, el bot decide qué situación pasada se asemeja más a la actual.
- El bot actúa igual que lo hizo en la situación anterior.
- Se determina si el resultado de esa situación ha sido satisfactorio o no.
- El valor de satisfacción para dicha situación se almacena en la base de casos.

De esta manera se van reorganizando los casos, hasta el punto en que para cada situación del entorno se tiene identificado el comportamiento más idóneo, por medio del valor de satisfacción.

Para alcanzar dicho objetivo se partirá del entorno de visualización que proporciona el juego Unreal Tournament 2004, juego de código abierto y con un conjunto de librerías que permiten crear

bots de manera independiente al entorno gráfico. Dicho conjunto de librerías se llaman Pogamut[1] (versión 3.1). La base de casos se creará con un simple gestor de bases de datos SQLite3[2], mediante su driver para java SQLiteJDBC. El uso de java se debe a que las librerías Pogamut se encuentran en dicho lenguaje. La forma de usar estas herramientas se comenta en la sección 2.1.

Como trabajo previo al planteamiento de la hipótesis se tiene que demostrar el correcto funcionamiento de las librerías Pogamut en el entorno del juego.

La hipótesis de este proyecto se centra en evaluar si se puede aplicar un sistema CBR en este tipo de juegos que requieren una rápida respuesta. La conclusión final determinará si es posible el uso de este sistema y cuáles son las mejores condiciones, para el aprovechamiento máximo y para que se obtengan los mejores resultados posibles. Como ya hemos indicado, se intentará obtener un patrón global que modele la estrategia del bot que mejor actúe en cualquier tipo de entorno del juego. Con este patrón global se quiere establecer de manera genérica cómo se debería afrontar un juego de estas características con la mayor probabilidad de éxito en el menor tiempo posible, evitando así que un jugador humano tenga que descubrir por él mismo cúal es la mejor forma de actuar.

En el Anexo A: "Planificación" se muestra el detalle temporal del proyecto.

1.2. Contenido de la memoria

A lo largo de este documento, se pueden encontrar las siguientes secciones:

- Introducción: Incluye los objetivos y se establece el contexto en el que se encuadra el trabajo.
- Cuestiones exploradas o trabajo previo: Cuestiones y herramientas consultadas para el desarrollo del proyecto.
- Descripción del modelo: Una vez establecidas las cuestiones y comprendidas las herramientas se explica el desarrollo del proyecto, detallando todo el proceso seguido para la realización del bot.
- Diseño de experimentos y resultados: Como resultado final del proyecto se establecen unos experimentos para comprobar que se cumplen los objetivos planteados.
- Conclusiones y valoración personal: Conclusión global sobre el desarrollo del proyecto, posibles líneas futuras y valoración del proyecto.
- Glosario: Definición de algunos términos.
- Bibliografía: Fuentes de información consultadas para realizar este proyecto.
- Anexos: Información adicional del proyecto, referenciada desde apartados anteriores.

1.3. Contexto

¿Pueden pensar las máquinas? Esta pregunta fue planteada por Alan M. Turing[3] en un famoso trabajo de 1950 [1] (Figura 1.1). Mediante un famoso test que lleva su nombre, Turing introduce un criterio operacional para dirimir si el comportamiento de un artefacto debería ser considerado

[1] http://diana.ms.mff.cuni.cz/main/tiki-index.php
[2] http://www.sqlite.org/
[3] Alan M. Turing, matemático, criptógrafo y filósofo inglés de mediados del siglo XX.

como “inteligente”. En principio parecería obvio que este don solo se puede asociar a los seres humanos, que durante millones de años han desarrollado las estructuras neuronales que conforman sus complejos cerebros. Sin embargo, la respuesta a esta pregunta no es sencilla. Dependerá, como bien dijo Turing, “de qué entendamos por pensar”. Según la Real Academia Española de la Lengua[4], “pensar” supone el acto de “examinar con cuidado algo para formar dictamen”. Una definición de este tipo deja una puerta abierta a la existencia de máquinas inteligentes.

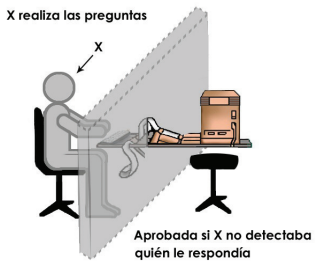


Figura 1.1: Explicación gráfica del test de Turing

En este trabajo nos interesamos por una de las versiones del Test de Turing aplicada al entorno de los videojuegos. Los videojuegos actuales, poseen una gran diversidad, y proporcionan un abanico de opciones capaz de satisfacer a cualquier tipo de público. Los principales géneros de videojuegos son: aventuras, deportes, educativos, estrategia, shooter, lucha, juegos de mesa, party, plataformas, rol y simulación (Ver Glosario). De todos estos géneros, este estudio se centrará en los de tipo Shooter, debido al concurso Botprize[5]. La finalidad de dicho concurso es crear bots para el Unreal Tournament 2004[6] que simulen el comportamiento de un jugador humano (en el Anexo B: “Botprize” se muestran las bases de la competición). Esta competición responde a una inquietud por parte de la industria del videojuego en incorporar herramientas “inteligentes” a sus diseños.

En estos últimos tiempos se han extendido las técnicas de inteligencia artificial de tal modo que actualmente cubren un conjunto de intereses más amplios que los iniciales. Una de las ideas que ha fomentado este cambio hacia una “nueva IA” es la de reproducir emociones humanas, algo impensable en la “IA tradicional”. Se considera que reproducir aspectos como las emociones o la personalidad de los seres humanos nos llevará a mostrar “humanidad” en las máquinas. Actualmente términos como emociones, conciencia y humanidad ocupan gran cantidad de proyectos dedicados a avanzar en técnicas de inteligencia artificial.

1.3.1. Emociones y videojuegos

Las emociones son fenómenos fisiológicos que nos hacen “sentir” y de las que podemos aprender para formar conductas que en el futuro nos ayudarán. Los videojuegos actuales tratan de dos modos distintos su relación con las emociones. Por una parte, buscan reforzar las conductas del jugador en diversos aspectos, por ejemplo, “enseñando a un niño a como razonar”. Por otro lado intentan despertar las emociones del jugador en momentos puntuales del juego para tener una mayor interacción. Existen muchas hipótesis o ideas que giran en torno a las emociones, restando importancia a la lógica. Una de las más importantes es la que se va a aplicar en este proyecto: La hipótesis del “marcador somático”.

[4] www.rae.es/rae.html
[5] http://botprize.org/
[6] http://www.unrealtournament.com/

Hipótesis del marcador somático

Antonio Damasio[7] en su libro "El error de descartes" [13] intenta demostrar que las emociones forman parte del razonamiento, lo que sería según él mismo dice, "una racionalidad de las emociones". Damasio intenta dar explicación a los comportamientos de algunos de sus pacientes que han sufrido daños en el lóbulo central. Éstos no muestran ningún signo de emoción ante situaciones sensibles o trágicas[8], y aunque conservan todas sus facultades cognitivas, pierden la capacidad para tomar decisiones. Damasio llega a la conclusión de que "la defectuosa capacidad de toma de decisiones se debe a la ausencia de emociones", registrando evidencias experimentales acerca de comportamientos nada exitosos por parte de estos sujetos, en dos aspectos principalmente:

- Emplean, irracionalmente, cantidades de tiempo desmedidas en tareas triviales[9].
- Toman malas decisiones – en vez de retrasarlas – por la falta de motivación que manifiestan sus representaciones mentales sobre estados futuros.

Damasio desarrolla el modelo de "marcadores somáticos" para explicar, desde un punto de vista neurofisiológico, el porqué del comportamiento de sus pacientes.

Su "hipótesis del marcador somático" propone que la participación emocional en el proceso de deliberación práctica y toma de decisiones, lejos de constituir una interferencia perturbadora, es más bien una condición de posibilidad del mismo. Si nuestra deliberación práctica tuviese que desarrollarse según los criterios de la teoría de la decisión[10], no podríamos tomar decisiones adecuadas en situaciones de urgencia, debido a la complejidad del cálculo requerido y al gran número de cursos de acción alternativos a evaluar. Este número, sin embargo, es drásticamente reducido, según Damasio, por la participación de las emociones en la selección de alternativas que serán tenidas en cuenta.

La idea de Damasio, de que las emociones ayudan a realizar una selección previa, ha generado una corriente de investigaciones que busca la manera más eficiente de modelar las emociones artificialmente. Estas investigaciones se centran en situaciones en las que la racionalidad no funciona tan rápido como se desearía [15] o en situaciones en las que no es necesario racionalizar completamente para tomar un decisión [16]. Para aplicar esta hipótesis existen técnicas que se ajustan perfectamente, como es el caso de los sistemas de razonamiento basados en casos, que pueden usarse para ver como las emociones reducen las opciones antes de tomar una decisión.

En el Anexo C: "Teoria de Damasio", se explica con más detalle su hipótesis del marcador somático, así como el modelo emocional que plantea.

Razonamiento Basado en Casos y Emociones

Kolodner[11] propuso en la década de los 80 un modelo de razonamiento conocido como "sistema de razonamiento basado en casos" [4]. Este modelado se caracterizaba por superar alguno de los

[7] Antonio Damasio, profesor de Neurociencia, Neurología y Psicología en la Universidad de Southern California y director del Instituto del Cerebro y la Creatividad.

[8] En [13] los experimentos realizados con este tipo de sujetos no presentan las respuestas de conductancia cutánea de los individuos normales.

[9] En [13] se narra la conducta de un paciente con el que Damasio pretende concretar una futura cita, y al que le sugiere dos fechas diferentes con sólo unos pocos días de separación. En palabras del propio Damasio:

[. . .] el paciente saco su agenda y empezó a consultar el calendario. [. . .] Durante casi media hora, el paciente enumeró motivos a favor y en contra de cada una de las dos fechas [. . .] llevado por un agotador análisis de costes y beneficios, [...] Finalmente le dijimos que debía venir la segunda de las dos fechas. Su respuesta fue igualmente tranquila y rápida. Dijo: <<Está bien>>.

[10] Según la teoría de decisión clásica [14] el mecanismo para la correcta elección de una opción estaría basado en un análisis lógico de la situación (sin contaminación emocional), se seleccionaría la opción que maximizase el provecho subjetivo esperado.

[11] Janet L. Kolodner es una científica cognitive Americana, profesora en el instituto tecnológico de Georgia.

problemas que afectaban a los sistemas expertos. En lugar de basarse exclusivamente en el conocimiento general del dominio de un problema, se utiliza el conocimiento específico de experiencias previas en situaciones concretas. Un sistema de razonamiento basado en casos fundamentalmente resuelve un problema, por medio de la adaptación de soluciones dadas con anterioridad a problemas similares [5].

Un sistema CBR típico, está compuesto por cuatro etapas secuenciales que se invocan siempre que es necesario resolver un problema [6]. La Figura 1.2, muestra el ciclo de vida de un sistema de razonamiento basado en casos clásico, donde las cuatro etapas del proceso son:

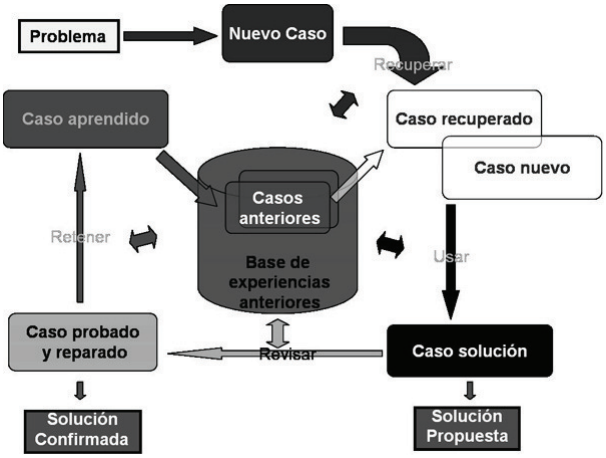


Figura 1.2: Ciclo de vida de un sistema CBR.

- Recuperación de los casos o problemas más relevantes al nuevo problema.
- Utilización (adaptación) de los casos o problemas recuperados con la intención de solucionar el problema presente.
- Revisión de la solución propuesta inicialmente por el sistema.
- Retención (aprendizaje) de la nueva solución como parte de un nuevo caso.

Al igual que otros mecanismos de resolución de problemas, el objeto de un sistema CBR es el de encontrar la solución a un problema determinado. La misión del algoritmo de recuperación, consiste en buscar en la memoria del sistema y seleccionar aquellos ejemplos más similares (junto con sus soluciones) al problema presente. Los casos (problema-solución) seleccionados son reutilizados para generar una solución. Si es posible, la solución se revisa y finalmente se crea un nuevo caso que se almacena en la memoria del sistema. Los casos almacenados en la memoria pueden ser eliminados o modificados, pudiendo crearse nuevos casos a través de la mezcla de características de otros ya existentes.

La intervención humana puede ser necesaria a lo largo del ciclo de vida de un sistema CBR, especialmente en las dos últimas fases del ciclo (revisión y retención). Este hecho supone uno de los mayores inconvenientes de esta metodología, y ha sido señalado como un gran inconveniente por sus detractores. En ocasiones se elimina esta fase para la automatización completa del sistema.

En el Anexo D: “Técnicas de Razonamiento basado en Casos” se amplía la explicación de los sistemas CBR.

La aplicación de sistemas CBR a videojuegos está empezando a dar sus frutos a la hora de crear comportamientos para NPC (Ver Glosario) más realistas. Principalmente se están haciendo

investigaciones en juegos RTS (Ver Glosario), ya que consiguen aprovechar mucho mejor el ciclo de vida de un sistema CBR. El mayor problema que tenían las técnicas tradicionales de inteligencia artificial con este tipo de juegos era su amplio espacio de búsqueda para decidir qué acción tomar en un momento dado. Con los sistemas CBR se consigue reducir este problema al extraer de jugadores expertos sus conocimientos y convertirlos en casos. Algunos de los juegos en los que han sido probadas estas técnicas (la mayoría son juegos RTS) son WARGUS [7, 8, 9], C-evo [10], RoboCup (no es un videojuego en sí mismo)[11] y Monopoly [12].

Capítulo 2

Cuestiones exploradas o trabajo previo

En esta sección se van a comentar los primeros pasos e importantes pasos de la realización de este proyecto, que se han dedicado a la investigación y a conocer el funcionamiento las herramientas que se van a usar, como son Pogamut y SQLite3. Se añadirá un breve apartado en el que se verán reflejadas las técnicas y herramientas que durante el desarrollo han sido consideradas para su posible utilización pero que por diversos motivos se fueron desechando.

2.1. Pogamut

Es una plataforma de código abierto usada para el rápido desarrollo de comportamientos en agentes virtuales incrustados en un entorno 3D del videojuego Unreal Tournament 2004. En [17] se puede encontrar una explicación más profunda de cómo funciona internamente esta plataforma, así como algunas de las técnicas implementadas por los creadores de Pogamut (ALMA[1] [18]). Entre otras cosas se explica la arquitectura de Pogamut y todos los componentes que la integran (Figura 2.1):

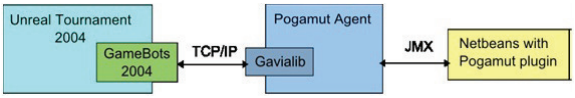


Figura 2.1: Arquitectura de Pogamut.

- **Unreal Tournament 2004**: Videojuego sobre el que se trabaja.
- **GameBots 2004**: Exporta e importa información del juego en lenguaje UnrealScript, a través de protocolos TCP/IP (Ver Glosario).
- **GaviaLib**: Librería dentro de Pogamut que se encarga de traducir UnrealScript (Ver Glosario) y pasarlo a Java[2] y permite conectar agentes a casi cualquier entorno virtual. En la Figura 2.2 se ve su arquitectura a alto nivel. (1) Procesa los mensajes del entorno. (2) Envía los comandos al mundo virtual. (3) Interfaz del agente.

[1] http://www.dfki.de/~gebhard/alma/index.html
[2] http://www.java.com/es/

- **Pogamut Agent**: Conjunto de clases java derivadas de las clases básicas de GaviaLib.
- **IDE Netbeans**[3]: Mediante su plugin de Pogamut permite trabajar con las librerías más cómodamente e incluso permite una fácil conexión con el servidor del juego.

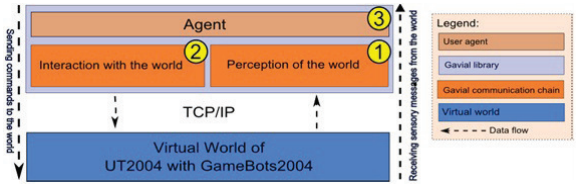


Figura 2.2: Arquitectura a alto nivel de GaviaLib

Antes de pasar a crear un bot hay que instalar la plataforma Pogamut y conectar el bot desde Netbeans al juego mediante los servidores UT2004 específicos del juego. Todo ello se explica con detalle en el Anexo E: "Instalación del entorno".

2.1.1. Cómo crear un bot

Una vez que se tiene el servidor conectado a Netbeans es el momento de empezar a programar el bot. Para eso lo primero es entender que proporciona Netbeans con su plantilla básica, y a partir de ahí desarrollar el bot.

La plataforma Pogamut tiene un sinfín de clases y métodos, pero todas están relacionadas por lo que conociendo un grupo seleccionado de clases se puede crear un bot y esas clases se encargan de comunicarse con el resto. Las clases principales son:

- cz.cuni.amis.pogamut.ut2004.utils.**UT2004BotRunner**: Lanza un thread de ejecución (Ver Glosario) para ejecutar al bot. Se sitúa dentro del main principal. Se acaba cuando se elimina al bot de la partida.
- cz.cuni.amis.utils.exception.**PogamutException**: En todas las funciones donde se ejecutan instrucciones de Pogamut hay que capturar excepciones. Se encarga de tratar cualquier excepción de Pogamut.
- cz.cuni.amis.pogamut.ut2004.bot.impl.**UT2004BotModuleController**: Contiene el controlador más avanzado del sistema. Tiene todos los módulos útiles para manejar el bot. Es la clase principal.
- cz.cuni.amis.pogamut.base.communication.**worldview**: Ofrece tanto los sentidos del bot y como la memoria simple. Muestra cómo está representado el mundo por medio de "listeners", eventos que van ocurriendo en el mundo. Esta clase es la encargada de recogerlos y tratarlos posteriormente si se quiere configurar el bot para que actue cuando los vaya recibiendo.

Además hay dos grupos de clases que son muy importantes ya que contienen los comandos y mensajes que se pueden introducir en GameBots2004, pero que no han sido incluidos como clases principales porque son las clases anteriores las que se encargan de gestionar esos comandos y mensajes (Anexo F: "Lista de comandos y mensajes" muestra una lista de todos los comandos y mensajes que existen en la comunicación entre UT2004, GameBots2004 y Pogamut).

A continuación se presenta en detalle el concepto de las dos clases más importantes.

[3]http://netbeans.org/

UT2004BotModuleController

Esta es la clase principal del sistema, la que se encarga de manejar los módulos que controlan al bot y de establecer el protocolo de comunicación (Ver Glosario) con el entorno. Las funciones que establecen el protocolo de comunicación son las siguientes:

- **prepareBot()**: Establece la configuración del mundo antes de saber nada sobre UT2004. Simplemente prepara el sistema virtual. Se ejecuta antes de conectar al bot con el entorno pero después de construir el UT2004Bot.
- **getInitializeCommand()**: Aquí es donde se caracteriza al bot con las opciones deseadas (nombre, habilidad, skin...). Envía un objeto `Initialize` que contiene todos los parámetros generales del bot.
- **botInitialized(GameInfo info, ConfigChange config, InitedMessage init)**: Primera vez que el bot tiene información sobre el mundo, aunque todavía no ha sido creado. Se llama a esta función una vez que el servidor ha enviado el mensaje INITED (clase de java: `InitedMessage`). Esto significa que el comando INIT ha sido ejecutado correctamente y la comunicación entre el bot y el servidor es correcta. En este momento ya es posible establecer nuevas comunicaciones entre el bot y el servidor, aunque todavía el bot no ha sido creado en el entorno, por lo que los comandos de movimiento no se pueden ejecutar, pero si los de información del juego (`GameInfo`), configuración del sistema actual (`ConfigChange`) y mensajes de inicialización (`InitedMessage`).
- **botSpawned(GameInfo info, ConfigChange config, InitedMessage init, Self self)**: Se ejecuta cuando se crea el bot por primera vez, justo al entrar en el entorno del juego.
- **logic()**: Función principal del sistema. Entra justo después de crear el bot. Solo sale de ella cuando tiene lugar algún evento externo (por ejemplo, se muere el bot...) o cuando se elimina al bot de la partida. Todo lo que se ejecute dentro de la función debe ser procesado rápidamente ya que se ejecuta cada 0.25 segundos.

Esas son las funciones que sirven para crear el protocolo de comunicación y este es el protocolo de comunicación:

1. Se invoca a `this.prepareBot()`
2. →Recibe un mensaje HELLO_BOT que se encarga de pedir la conexión de un bot con el servidor. Si el servidor está saturado u ocupado se termina la conexión.
3. ←Envía un mensaje READY. Como respuesta el servidor envía al juego un mensaje NFO (info) con información sobre la partida.
4. Ahora es cuando se establece la comunicación con GameBots2004.
5. →Captura el evento `InitCommandRequested` confirmando que la inicialización está preparada.
6. Ejecuta `this.getInitializeCommand()`.
7. ←Envía el comando INIT que se ha creado en la función anterior, con los parámetros iniciales del bot.
8. →Recibe el mensaje `ConfigChange`.
9. →Recibe el mensaje `InitedMessage`.

10. Invoca a `this.botInitialized()`.

11. ... se recibe el primer mensaje SLF (self) mostrando la información que el bot ya puede conocer de sí mismo, al haberse introducido en el sistema. También recibe el mensaje END para terminar la comunicación síncrona con el servidor.

12. Ejecuta `this.botSpawned()`.

13. El bot pasa a ejecutarse dentro del entorno por medio de la función `logic()`.

Después de establecer el protocolo de comunicación del bot con el entorno hay que conocer todo lo que el bot puede hacer, para eso existen los módulos del bot y los eventos o listeners. Los módulos principales son los siguientes:

- **Game**: Obtiene información general sobre el juego. Con este módulo se puede saber: el tipo de juego (DeathMatch, Capturar bandera, etc), el nombre del mapa, tiempo total de la partida, número máximo de equipos y salud, armadura y adrenalina que dispondrá el bot al inicio y la máxima que puede llegar a conseguir.

- **Info**: Información sobre el paradero del agente. Con este módulo podemos saber todo sobre el bot: su nombre y su equipo, qué jugadores y equipos son enemigos y quiénes son amigos, localización, velocidad actual, posición en la que se encuentra, salud, armadura y adrenalina actual, qué arma usa...

- **Players**: Informa sobre otros jugadores. Todos los objetos de este tipo se auto-actualizan a lo largo del tiempo hasta que son destruidos. Este módulo permite al bot conocer todo esto sobre el resto de jugadores: informar sobre la posición, la visibilidad...

- **Items**: Objetos que se encuentran en el mapa. Se puede obtener: Mapas con todos los items del escenario (se puede restringir la búsqueda a los objetos visibles o a los alcanzables), obtener un único ítem...

- **Senses**: Parte sensorial del bot. Se puede controlar todo lo que afecta al bot: Informar donde fue golpeado, último elemento recogido, último daño que ha causado, último daño que le han hecho, informar sobre la muerte de un jugador...

- **Weaponry**: Inventario del bot (armas y munición). El módulo le permite al bot saber: Cantidad de munición del arma actual (disparo primario y secundario) o de todas las armas del inventario, características del arma que está usando...

- **RayCasting**: Es una de las opciones que dispone el bot para moverse por el mundo (las otras son por puntos de navegación o por localizaciones). Crea rayos que permiten detectar objetos en la escena. El rayo es verde cuando el camino está libre y rojo cuando hay obstáculos en la dirección y longitud del rayo. Lo que indican los rayos es que el bot circule solo por las zonas donde los rayos son verdes.

- **Shoot**: Son acciones de disparo avanzadas que puede realizar el bot: Disparar arma primaria o secundaria a un objetivo o a una localización, recargar arma primaria o secundaria y parar de disparar.

- **Move**: Otra de las opciones de movimiento. Puede mezclarse con las otras dos sin ningún problema. Son movimientos avanzados que puede realizar el bot: Esquivar disparos u objetivos, saltar, ir a una localización determinada, correr, parar el bot repentinamente, poner el foco de atención en un objeto o jugador y rotar sobre sí mismo.

- **PathPlanner**: Junto con `pathExecutor` es la tercera forma de moverse. Se encarga de calcular la ruta de acceso a un punto usando como apoyo los puntos de navegación del mapa. A partir de dos puntos del mapa es responsable de encontrar la ruta genérica. Hay varias formas de implementar esta ruta, como pueden ser el algoritmo de Dijkstra[4], A*[5]... (el método A* ya esta implementado dentro de pogamut). Aun así no es necesario seguir este método genérico incluido dentro de pogamut.

- **PathExecutor**: Se encarga de ejecutar el `pathPlanner` que haya sido calculado previamente. Cuando ha llegado a uno de los puntos del path vuelve a pedir el `pathPlanner` por si ha sido actualizado mientras estaba en ejecución. Con ese nuevo path vuelve a moverse hasta el siguiente punto y así hasta completar todo el `pathPlanner` calculado.

- **ListenerRegistrator**: Es el método encargado de autoinicializar los listeners por medio de `AnnotationListenerRegistrator`. Se encarga de proporcionar una manera sencilla y práctica de registrar los listeners. Todo este mundo de los eventos se explica en el siguiente punto de este apartado (WorldView).

WorldView

Esta clase y sus derivadas se encargan de recoger la información del entorno del juego. La mayor parte de la información recogida es procesada directamente por los módulos comentados en el punto anterior, pero aun así el propio usuario puede capturar los listeners (eventos) que van surgiendo y actuar en función del evento recogido.

El entorno está representado por objetos y eventos. Dichos eventos pueden estar asociados a objetos o no.

Hay 18 tipos de objetos (Tabla 2.1) que podemos encontrarnos en el mundo, pero todos ellos tienen una función o un comando que se encarga de tratarlos.

AliveMessage	FlagInfo	ItemCategory	Self
AutoTraceRay	GameInfo	Mover	TeamScore
BombInfo	IncomingProjectile	MyInventory	Vehicle
ConfigChange	InitedMessage	NavPoint	DominationPoint
Item	Player		

Tabla 2.1: Tipos de objetos en UT2004

Con los eventos no pasa lo mismo, no todos los eventos están controlados por comandos o funciones.

Los eventos se pueden dividir en dos: asociados y no asociados a objetos. Dentro de los eventos asociados a objetos hay 5 tipos:

- **WorldObjectAppearedEvent**: Avisa cuando aparece un objeto determinado.

- **WorldObjectDestroyedEvent**: Avisa cuando un objeto determinado es destruido.

- **WorldObjectDisappearedEvent**: Avisa cuando un objeto determinado desparece.

- **WorldObjectFirstEncounteredEvent**: Avisa cuando un objeto determinado se encuentra por primera vez.

[4] http://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra
[5] http://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*

- **WorldObjectUpdatedEvent**: Avisa cuando un objeto determina actualiza su situación (player).

Entre los eventos no asociados a ningún objeto se incluyen mayoritariamente eventos que son utilizados para la conexión con el servidor, lo cual ya está implementado y no se necesitan tratar. También hay eventos que indican el principio y el final de la llegada de un lote síncrono de datos por lo que tampoco son necesarios. La lista completa se encuentra en el Anexo G: "Listeners", pero a continuación se muestran los eventos más importantes a tratar:

- **AddInventoryMessage**: Consigue nueva arma, o munición para un arma que todavía no tiene.
- **BotDamaged**: El bot ha sido dañado.
- **BotKilled**: El bot ha muerto.
- **FallEdge**: El bot está al borde de un precipicio. Si el bot estaba corriendo, el mensaje llega cuando el bot está cayendo.
- **HearNoise**: Avisa cuando oye un sonido. Puede ser un jugador caminando o disparando, una bala impactando contra el suelo, un ascensor subiendo y bajando, etc.
- **PlayerDamaged**: El bot hiere a otro jugador.
- **PlayerKilled**: Otro jugador ha muerto.
- **VolumeChanged**: Alguna parte del bot ha cambiado de zona (al agua, lava, etc.).
- **WallColission**: El bot ha colisionado contra un muro.

Cuando se quiere utilizar uno de estos eventos u objetos hay que saber cómo manipularlos, es decir, cuál es la forma más sencilla de recibir dicha información.

La clase `UT2004BotModuleController` los auto-inicializa con `ListenerRegistrator`. Esto quiere decir que no será necesario manipular los listeners manualmente, solo hay que registrarlos.

Hay cinco diferentes tipos de listeners que se pueden registrar, pero los más útiles son:

- **EventListener**: reacciona al evento que nosotros definamos por medio del campo eventClass. Por ejemplo: `@EventListener(eventClass = Bumped.class)`, reacciona a eventos de la clase Bumped.
- **ObjectClassListener**: reacciona ante todos los eventos ocurridos a una clase de objeto concreta, definida por medio del campo objectClass. Por ejemplo: `@ObjectClassListener (objectClass = Player.class)`, reacciona a eventos ocurridos a la clase Player.
- **ObjectClassEventListener**: reacciona a eventos de una clase concreta (de entre los eventos asociados a objetos) que le ocurren a objetos de una clase determinada. Por ejemplo: `@ObjectClassEventListener(eventClass = WorldObjectAppearedEvent.class, objectClass = Player.class)`, reacciona cuando un jugador cualquiera "aparece" en nuestro campo de visión.

2.2. SQLite

SQLite es un sistema de gestión de bases de datos relaciones.

No es como una base de datos cliente-servidor (Ver Glosario), en SQLite el motor se enlaza con el programa pasando a ser parte del mismo, de esta manera reduce la latencia (Ver Glosario) de acceso a la base de datos, debido a que las llamadas simples a subrutinas o funciones son más eficientes que la comunicación entre procesos (técnica usada por las bases de datos cliente-servidor).

El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

La decisión de usar este tipo de librerías como base de datos se ve reflejada en el tiempo de latencia antes comentado, que es más rápido que en otros tipos de base de datos. Esto es importante ya que para que el ciclo de ejecución establecido por Pogamut de 0.25 segundos no se sobrepase, los accesos deben ser lo más rápidos posibles.

2.3. Técnicas y herramientas desechadas

A continuación se van a comentar brevemente algunas de las técnicas, herramientas o proyectos anteriores que se han consultado para su posible utilización, pero por diversos motivos no han si usados para el posterior desarrollo del bot.

- **ALMA**[6]: Es un modelo para el manejo de emociones, que unido a Pogamut intenta dar un comportamiento emocional al bot dividiendo su comportamiento en emociones, estados de humor y personalidad. Aunque la idea parecía interesante no se consideró como un camino a seguir.
- **Posh**[7]: Otra de las técnicas consultadas para determinar los comportamientos. En este caso POSH (Parallel-rooted, Ordered Slip-stack Hierarchy) es una arquitectura que interpreta planes dinámicos (planificación reactiva). En este caso se desechó su uso porque parecía más interesante el uso de árboles a la hora de controlar el comportamiento del bot en vez de usar planes de rutas que es lo que propone este sistema.
- **jColibri2**[8]: Es un framework (Ver Glosario) orientado a objetos que facilita la construcción de sistemas de razonamiento basado en casos (CBR). Principalmente proporciona el esqueleto de un sistema CBR para poder integrarlo en cualquier aplicación que se quiera crear basada en un sistema CBR. El problema de usar este framework es que el sistema compuesto por UT2004, GameBots2004 y Pogamut obliga a modificar el concepto clásico de un CBR y adaptarlo a las posibilidades ofrecidas.

[6] http://www.dfki.de/~gebhard/alma/
[7] http://www.cs.bath.ac.uk/~jjb/web/posh.html
[8] http://gaia.fdi.ucm.es/projects/jcolibri/

Capítulo 3

Descripción del modelo

Después de ver cómo es el entorno en que se trabaja y cómo se puede crear cualquier bot, en esta sección se van a comentar los aspectos concretos relacionados con la creación del SomaticMarkerBot, bot basado en un sistema CBR para modelar la hipótesis de marcadores somáticos, se definirán las condiciones que va a tener la partida, la base de casos, las variables de entorno elegidas para controlar el bot, las funciones que hacen moverse al bot y los comportamientos que tiene asociados, en resumen la descripción del modelo utilizado.

3.1. Condiciones de la partida

Aunque no es algo que forme parte del desarrollo del bot y sí de las pruebas que posteriormente se realizan, lo primero antes de empezar a crear el bot, es constatar cuáles van a ser las condiciones generales de la partida sobre la que se van a hacer dichas pruebas o experimentos. En la Tabla 3.1 se muestran los 8 niveles que pueden tener los rivales, en la Tabla 3.2 los 29 tipos de mutadores (Ver Glosario) existentes y en la Tabla 3.3 todas las condiciones que se pueden configurar a la hora de crear el servidor. Es importante definir correctamente estas condiciones porque afectan al desarrollo de las pruebas y, en especial, afectan al comportamiento del bot.

1. Novato	2. Aventajado	3. Experimentado	4. Habilidoso
5. Adicto	6. Maestro	7. Inhumano	8. Invencible

Tabla 3.1: Nivel de experiencia de los bot UT2004.

AirControl	MutBonusVehicles	Super Berserk
Arena	No Adrenaline	Tranquilliser Guns
BigHead	No SuperWeapons	UDamage Rewards
Bonus Combos	Onslaught Weapons	UT Classis
GBHUDMutator	PathMarkerMutator	UT2003 Style
GBHUDMutator	QuadJump	UTV2004S
GBNoWeaponMutator	Regeneration	Vampire
InstaGib	Slow Motion Corpses	Vehicle Pickups
Lightning Guns	Sniper Rifles	Zoom InstaGib
LowGrav	Species Statistics	

Tabla 3.2: Mutadores del juego.

CONDICIÓN	MÁXIMO	ELEGIDO	¿POR QUÉ?
Número de participantes	32	4	Suficiente número de participantes para no sobrecargar el sistema y para tener siempre enemigos visibles.
Niveles del bot	8	3	Nivel "jugador experimentado", es un nivel adecuado a la mayoría de jugadores típicos del juego.
Autoajustar nivel	SI/NO	NO	Para los resultados será mejor no tener unos enemigos que varíen su calidad.
Permiso de recoger armas	SI/NO	SI	Cualquier arma del escenario puede ser recogida.
Puntos objetivo	999	25	Menos de 15 puntos son partidas muy rápidas y poco aclaratorias. Más de 40 son partidas redundantes.
Tiempo de juego	999 min	30 min	Para el número de puntos objetivo los 30 minutos son más que suficientes.
Protección al renacer	30.0 s	2.0 s	Tiempo invulnerable recién renacido (sino dispara). Da tiempo a un primer movimiento pero no permite coger armas con seguridad.
Color de los equipos	SI/NO	NO	En una partida DeathMatch no es necesario.
Arrojar armas	SI/NO	SI	Puede tirar el arma si interesa.
Agitar armas	SI/NO	SI	Los disparos del arma tienen vibraciones.
Translocador inicial	SI/NO	NO	Las únicas armas iniciales son la shield gun (para protegerse) y el Assault Rifle (para atacar).
Sarcasmo	SI/NO	NO	No es necesario.
Mutadores	29	0	Complica el nivel de la partida y las posibilidades, por lo que se prescinde de ellos, ya que en las partidas normales no existe ninguno.

Tabla 3.3: Parámetros globales de la partida.

Con estos datos ya se ha establecido una parte del entorno de trabajo. A continuación se mostrarán las variables de entorno elegidas para proporcionar los conocimientos que tendrá el bot sobre dicho entorno. Dichas variables, como se verá, van a reducirse drásticamente, porque el bot solo puede manejar un puñado de ellas pero el juego proporciona más de cien posibles variables, desde una simple información de salud del bot, hasta el tipo de suelo que está pisando

3.2. Variables de entorno

Las variables de entorno codifican toda la información que el bot puede recibir del juego y que le permiten saber cómo es la situación en la que se encuentra y a partir de esa situación saber

cómo actuar (buscando en su memoria un caso parecido).

Después de analizar todas las variables, se llegó a la conclusión de que las que usaría el bot para situarse y comparar casos serían las siguientes:

- **DeathsToWin**: Muertes que quedan para ganar la partida. Útil para saber si los comportamientos están siendo buenos o no (cuántas menos muertes queden más cerca estará de la victoria y mejor se estará comportando).
- **People**: Determina si hay rivales en su visión.
- **EndGame**: Tiempo que falta para terminar la partida. Es interesante saber si la partida se está alargando en exceso o no, para arriesgar más o menos.
- **Armor**: Cantidad de escudo que tiene el bot en ese instante. Es un control de estado del bot básico
- **Ammo**: Cantidad de munición que tiene el bot en ese instante sobre el total que puede conseguir para ese arma. Es un control de estado del bot básico
- **Health**: Cantidad de salud que tiene el bot en ese instante. Es un control de estado del bot básico.
- **MyDeaths**: Número de muertes sufridas por el bot. Puede ayudar a determinar si los comportamientos son muy agresivos o muy contemplativos.
- **Sensor**: Controla si ha muerto recientemente o si está viendo un proyectil en ese momento. Si ha muerto recientemente no tendrá armas y puede ser menos aconsejable participar en la batalla.
- **NWeapon**: Cantidad de armas en el inventario. Nunca es bueno ir a una batalla sin armas.
- **NAmmo**: Cantidad de munición en el inventario sobre el total que puede obtener. Nunca es bueno ir a una batalla sin munición.

En el Anexo H: "Variables de entorno" hay una extensa lista con todas las variables que proporciona el juego, explicando su significado y porqué no han sido elegidas. Los motivos principalmente se centran en que son informaciones de nivel avanzado y que el bot no necesita conocer para poder moverse por el entorno.

Una vez establecidas las condiciones de la partida y la información que podrá recibir, se debe establecer el bot.

3.3. SomaticMarkerBot

Al plantear la idea de usar CBR como la estructura para representar los marcadores somáticos en la Teoría de Damasio, en un juego como UT2004, lo más natural es encontrarse problemas que impidan desarrollar la metodología de manera exacta a como se plantea teóricamente, por eso mismo el sistema CBR propuesto en este proyecto varía sobre el original. La Figura 3.1 muestra las diferencias entre lo que se puede considerar un CBR clásico y lo que sería el CBR modificado para adaptarse al UT2004. En general no hay grandes diferencias conceptuales. Pasamos a indicar las modificaciones:

Los pasos que sigue este CBR modificado son los siguientes:

Antes de empezar, la base de casos se carga la memoria del bot (en forma de algunos casos). Cada caso consta de:

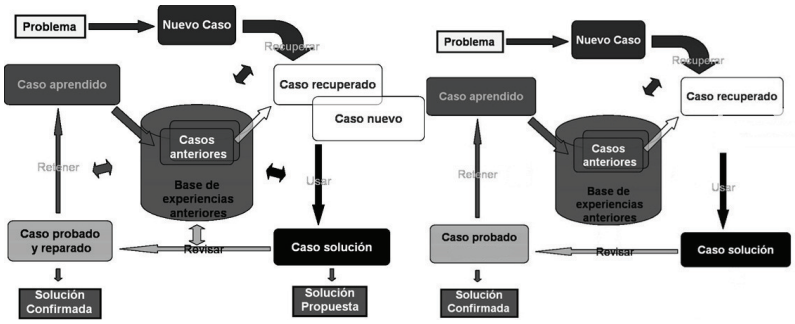


Figura 3.1: CBR clásico y CBR UT2004.

- **Variables de entorno**: Informan cómo está el juego en un momento determinado (sección 3.2).
- **Comportamiento**: Forma de actuar que tuvo el bot en ese momento. Se guarda una asignación al comportamiento, por lo que no es posible modificar los comportamientos en tiempo real.
- **Adaptabilidad**: Valor que simula el marcador somático de Damasio, indicando lo buena o mala que es la relación entre variables de entorno y comportamiento en un caso particular. Con este valor se puede simular el concepto de emoción a la hora de elegir una opción, tal como propone Damasio. Las parejas "variables de entorno-comportamiento" que hayan sido negativas para el bot en su ejecución bajarán su valor de adaptabilidad discriminándolas antes de tenerlas en cuenta en futuras situaciones. Y las que hayan sido positivas aumentarán su valor, estando siempre en la parte superior de la base de casos, consiguiendo que siempre el bot las tenga en cuenta, aunque luego, al deliberar, elija otra de menor valor pero más adecuada a la situación real.

Dicha base de casos debe permitir modificar, leer y eliminar casos. Dichas funciones están explicadas en el Anexo I: "Funciones de la base de casos".

Una vez el juego ha sido lanzado, la ejecución es iterativa hasta que acaba la partida. El proceso es el siguiente:

- **Nuevo Caso**: El bot recoge información del juego (variables de entorno elegidas) y crea un nuevo caso con valor de adaptabilidad y comportamiento nulos.
- **Recuperar**: Se busca en la base de casos aquel que más se parece al actual. Como ya se ha dicho antes estarán ordenados por el valor de adaptabilidad y sólo se tendrán en cuenta los primeros. De esta criba por adaptabilidad se pasa a la selección de un único caso, en función de la métrica, que más se asemeje a la situación actual.
- **Usar**: Una vez se tiene el caso recuperado, se ejecuta su comportamiento un número de veces suficiente para que haya podido probarse su utilidad. Con este se tiene el caso solución.
- **Revisar**: En este momento se debe comprobar cómo ha funcionado el caso. La métrica que comprueba la utilidad se basa en el número de muertes que ha sufrido el bot, el número de muertes que ha provocado y la variación de salud, todo en el periodo de tiempo en el que el caso ha estado ejecutándose. Con todo ello se actualiza el valor de adaptabilidad para ese caso.
- **Retener**: Finalmente se modifica la base de casos y vuelve a repetirse todo el proceso.

La principal diferencia con el CBR típico es que los casos son introducidos inicialmente, debido a problemas en la creación de comportamientos en tiempo real y solo se modifica el valor de adaptabilidad, en ningún caso se crea un nuevo caso mezcla de otros anteriores, ni se repara un caso creado, ya que no se modifican.

Al no permitir que los casos se creen en tiempo de ejecución, el bot necesita un amplio conjunto de comportamientos variados, para facilitar su adaptación a diversas situaciones y que esos mismos comportamientos vayan ordenandose según avanza el bot en la partida.

3.3.1. Comportamientos

Los comportamientos son los que permiten al bot adaptarse y actuar de diversas maneras (que pueden ser mejores o peores en función de la situación en la que se encuentre).

Los comportamientos se han creado en forma de árbol (Figura 3.2), por lo que dependiendo de las condiciones de la partida, el árbol binario[1] discurre por una rama u otra, actuando de la mejor manera posible. En las hojas de cada comportamiento están las llamadas "acciones simples". El tamaño de los árboles es variable, entre cuatro y cinco niveles, como se verá en el Anexo K: "Árboles de comportamiento".

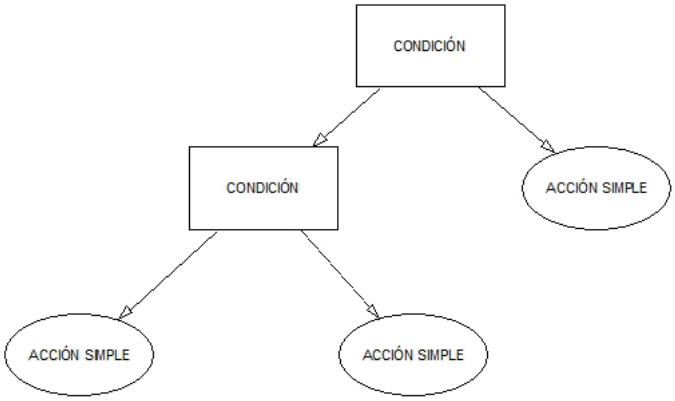


Figura 3.2: Árbol binario

Estas "acciones simples" son las que determinan qué va a hacer el bot (recoger objetos, atacar, huir...). Hay muchas posibilidades a la hora de elegir cuales van a ser esas acciones simples, tantas como se quiera, pero para el ámbito de este proyecto se determinó que había que restringirlas de la mejor manera posible y elegir las consideradas más útiles para el estudio, puesto que el proyecto se desarrolla en un entorno acotado. En el Anexo J: "Acciones simples" hay un completo y extenso análisis de cuáles son las mejores acciones simples y porqué han sido elegidas frente a otras opciones también consideradas. El resultado final de ese análisis se muestra en la Tabla 3.4.

Acción simple	**¿Que hace?**
walk	Permite andar por el mundo de un punto de navegación a otro.
protector	Recoge objetos de salud o de escudo que estén disponibles.
pickHealth	Recoge objetos de salud que estén disponibles.

[1] http://es.wikipedia.org/wiki/%C3%81rbol_binario

Acción simple	¿Que hace?
pickArmor	Recoge objetos de escudo que estén disponibles.
army	Recoge cualquier arma que esté disponible y cualquier munición disponible asociada a un arma del inventario del bot.
runAwayFromPlayer	Huye de un rival en dirección contraria siempre que pueda.
attackNearestVisiblePlayerWithSpeed	Ataca al jugador visible más cercano usando su arma más rápida.
attackNearestVisiblePlayerWithAmmo	Ataca al jugador visible más cercano usando su arma con más munición.
attackNearestVisiblePlayerWithDamage	Ataca al jugador visible más cercano usando su arma que provoque más daño.
attackNearestPlayerWithSpeed	Ataca al jugador más cercano usando su arma más rápida.
attackNearestPlayerWithAmmo	Ataca al jugador más cercano usando su arma con más munición.
attackNearestPlayerWithDamage	Ataca al jugador más cercano usando su arma que provoque más daño.
attackRandomVisiblePlayerWithAccuracy	Ataca a un jugador visible aleatoriamente usando su arma más precisa.
attackRandomVisiblePlayerWithDamage	Ataca a un jugador visible aleatoriamente usando su arma que provoque más daño.
attackFurthestVisiblePlayerWithAccuracy	Ataca al jugador visible más lejano usando su arma más precisa.
attackFewerDeathsPlayerWithSpeed	Ataca al jugador que tenga menos muertes usando su arma más rápida.
attackFewerDeathsPlayerWithAmmo	Ataca al jugador que tenga menos muertes usando su arma con más munición.
attackFewerDeathsPlayerWithDamage	Ataca al jugador que tenga menos muertes usando su arma que provoque más daño.
attackMostDeathsPlayerWithSpeed	Ataca al jugador que tenga más muertes usando su arma más rápida.
attackMostDeathsPlayerWithAccuracy	Ataca al jugador que tenga más muertes usando su arma más precisa.
attackMostDeathsPlayerWithDamage	Ataca al jugador que tenga más muertes usando su arma que provoque más daño.

Tabla 3.4: Acciones simples.

Una vez que se tienen las 21 acciones simples, que conformarán las hojas de los árboles de comportamiento, hay que determinar que comportamientos van a crearse. Con 21 acciones simples, las combinaciones posibles de comportamientos crecen exponencialmente por lo que en este caso la mejor opción era ir a buscar agrupaciones interesantes y variadas, ya que reducir de manera lógica todas las opciones obtenidas era impensable ($\sim 21 \ \sim 5 * 10^{19}$ sin contar con que la forma

de colocar las acciones, también puede modificar los resultados).

Finalmente se llegó a la conclusión de tener 2 formas diferentes de recolectar objetos para comprobar cuál de las dos era mejor (si recoger elementos de salud, escudo y armamento, o recoger elementos de salud/escudo y armamento) y 4 formas de ataque (para armas rápidas, armas con mucho daño, armas con mucha munición y una mezcla de ataques arriesgados, como, por ejemplo, atacar al más lejano o al que más muertes lleva asignadas). Además para tener mayor variedad de comportamientos se establecieron dos grupos de comportamientos, de ataque y defensivos, dentro de los cuales hay pequeñas variaciones, pero que no afectan al diseño del árbol, únicamente sirven para determinar si dentro de un comportamiento de ataque hay que ser más o menos agresivo, o si en un comportamiento defensivo hay que ser muy cauto o no es necesario serlo tanto. Un ejemplo de árbol de comportamiento es el de la Figura 3.3.

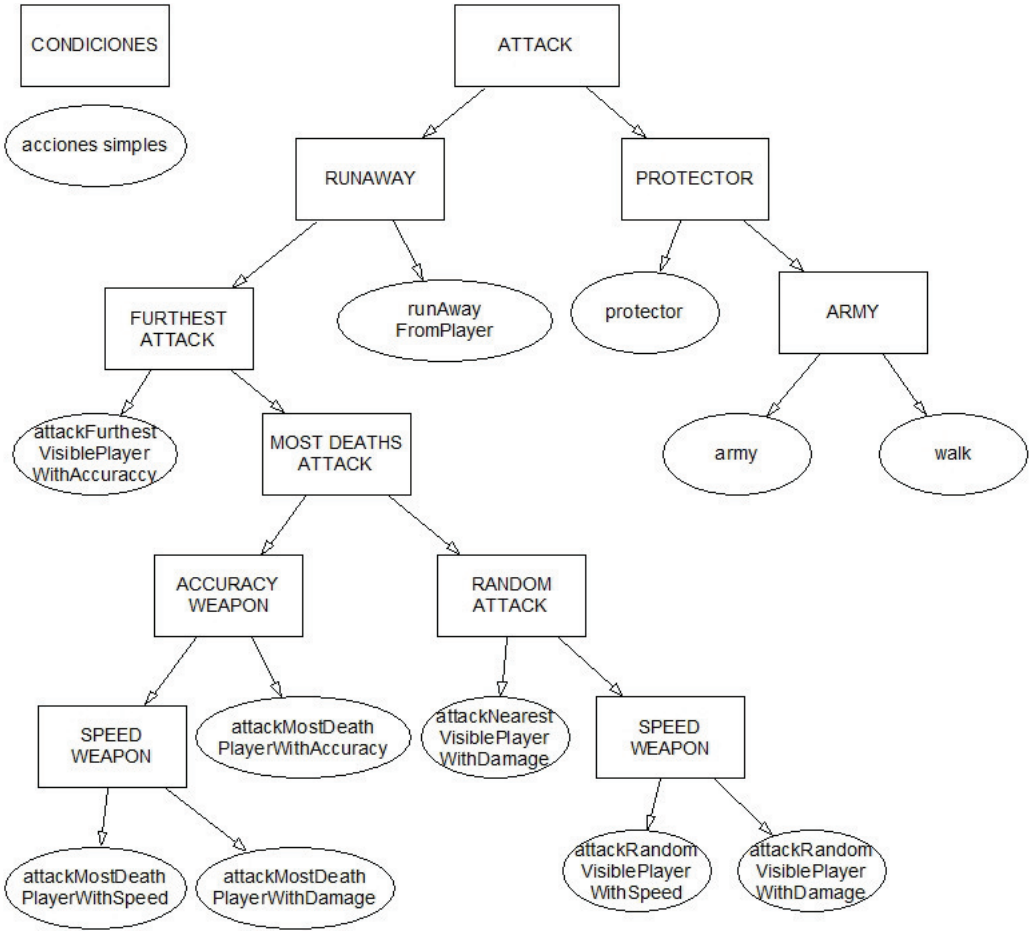


Figura 3.3: Árbol de comportamiento.

El conjunto de todos los árboles (16 en total) se encuentran en el Anexo "Árboles de comportamiento". En total si sumamos los diferentes comportamientos tenemos 16 posibilidades (56 si atendemos a las leves modificaciones dentro de cada grupo), que se reparten entre las 112 situaciones de entorno prefijadas para formar parte de la memoria del bot. Con esta cantidad de casos se asegura que exista una amplia gama de posibles soluciones (tanto defensivas, como de ataque) para entornos similares. Con todos estos elementos establecidos ya tenemos el SomaticMarkerBot creado y dispuesto a pasar por una batería de experimentos que intentarán demostrar si es útil usar un sistema CBR en videojuegos de respuesta rápida como UT2004.

Capítulo 4

Diseño de Experimentos y Resultados

Los experimentos se han desarrollado con dos objetivos en mente: (1) probar si las técnicas aplicadas en este proyecto son adecuadas para el manejo de bots y una vez conseguido, (2) obtener un patrón de comportamiento genérico que muestre la mejor estrategia de actuación de un bot independiente de las condiciones concretas en el entorno del videojuego utilizado.

En esta sección se dará una explicación de los parámetros que componen cada configuración experimental, se mostrarán los resultados de cada experimento discutiendo sus resultados y se establecerán unas conclusiones globales a partir de ellos. El objetivo es obtener un patrón de comportamiento universal para un bot cualquiera en el entorno del videojuego Unreal Tournament.

4.1. Parámetros de los experimentos

Para cada experimento se han establecido unas condiciones fijas de la partida (Tabla 3.3). Además hay tres condiciones que varían a lo largo de los experimentos y que son las que permiten obtener variedad de resultados. Estas tres condiciones son: la ventana de trabajo, la métrica de selección de caso y la métrica "premio/castigo".

4.1.1. Ventana

La ventana de trabajo simboliza los elementos de la base de casos que están visibles para el bot, o sea, en la ventana de trabajo se encontrarán los casos que los "marcadores somáticos" del bot hayan determinado como casos válidos. Como ya se ha visto, el marcador que utiliza el bot y que permite ordenar la lista de experiencias es el valor de adaptabilidad que "mide" si un caso ha sido bueno o malo previamente. En estos experimentos se va a trabajar con ventanas de 5, 25 y 50 casos.

4.1.2. Métricas de selección de caso

Se utilizan para elegir un único caso entre los situados en la ventana de trabajo. Después de seleccionar una serie de casos (ventana), esta métrica se encarga de determinar cuál de ellos es el más idóneo en la situación actual. Cuantos más parámetros se tengan en cuenta en esta métrica más discriminativa será la elección, ya que habrá más variedad en los valores de elección de caso. El más idóneo será aquel que difiera menos de la situación actual. Son utilizadas cuatro métricas.

Métrica 1

Tiene en cuenta a todas las variables de entorno que controla el bot. Cada variable de entorno (sección 3.2) del caso que se está probando se compara con el valor de la misma variable de entorno en el momento actual y se multiplica por un valor, que junto al rango de valores que puede tener cada variable de entorno, determina la importancia que tiene.

```
dtw = 10 * Δ (DeathsToWin_caso, DeathsToWin_actual).[1]
p = 45 * Δ (People_caso, People_actual).[2]
e = 3 * Δ (EndGame_caso, EndGame_actual).[3]
ar = 5 * Δ (Armor_caso, Armor_actual).[4]
am = 30 * Δ (Ammo_caso, Ammo_actual).[5]
he = 10 * Δ (Health_caso, Health_actual).[6]
my = 10 * Δ (MyDeaths_caso, MyDeaths_actual).[7]
s = 15 * Δ (Sensors_caso, Sensors_actual).[8]
nw = 3 * Δ (NWeapons_caso, NWeapons_actual).[9]
na = 30 * Δ (NAmmos_caso, NAmmos_actual).[10]
valor_idoneidad = dtw + p + e + ar + am + he + my + s + nw + na.
```

Métrica 2

Tiene en cuenta las muertes que quedan para ganar la partida (DeathsToWin). Premia matar enemigos lo antes posible.

```
valor_idoneidad = 10 * Δ (DeathsToWin_caso, DeathsToWin_actual).
```

Métrica 3

Tiene en cuenta las muertes que ha sufrido el bot (MyDeaths). Da preferencia a no sufrir muertes, antes que a matar enemigos.

```
valor_idoneidad = 10 * Δ (MyDeaths_caso, MyDeaths_actual).
```

Métrica 4

Tiene en cuenta las muertes que quedan para ganar (DeathsToWin), las muertes sufridas (MyDeaths) y la salud del bot (Health). Busca un equilibrio a las dos métricas anteriores.

```
dtw = 10 * Δ (DeathsToWin_caso, DeathsToWin_actual).
he = 10 * Δ (Health_caso, Health_actual).
my = 10 * Δ (MyDeaths_caso, MyDeaths_actual).
valor_idoneidad = dtw + he + my.
```

[1] [0, 250]
[2] [0, 45]
[3] [0, 180]
[4] [0, 150]
[5] [0, 30]
[6] [0, 1000]
[7] [0, 250]
[8] [0, 45]
[9] [0, 24]
[10] [0, 30]

4.1.3. Métricas "premio/castigo"

Se encargan de valorar si un comportamiento ha sido bueno o malo. Dependiendo del modo en que se premie o castigue al bot, cambia su modelo de aprendizaje. Con una mala métrica "premio/castigo" el bot no aprenderá satisfactoriamente y no logrará los objetivos. Solo se tienen en cuenta tres de las variables de entorno: DeathsToWin, MyDeaths, Health. Las métricas utilizadas son también tres.

Matar siempre

Premia matar cuantos más rivales mejor y castiga severamente no matar a nadie.

```
dea = Δ (MyDeaths_inicioComport, MyDeaths_finalComport).
kil = 1.2 * Δ (DeathsToWin_inicioComport, DeathsToWin_finalComport).
extra = -2 sino mata.
hea = (Δ (Health_inicioComport, Health_finalComport))/100.
NuevaAdaptabilidad = (Antigua - dea + kil + hea + extra)/10.
```

Evitar muerte, pero actuando

Premia al bot para que no le maten, pero sigue penalizando que no mate, aunque no tanto como la anterior.

```
dea = 1.2 * Δ (MyDeaths_inicioComport, MyDeaths_finalComport).
kil = Δ (DeathsToWin_inicioComport, DeathsToWin_finalComport).
extra = -0.2 sino mata.
hea = (Δ (Health_inicioComport, Health_finalComport))/100.
NuevaAdaptabilidad = (Antigua - dea + kil + hea + extra)/10.
```

Evitar muerte huyendo

Premia al bot para que no le maten, incluso evitando siempre que se pueda una confrontación. No matar a otro bot y que no le maten a uno mismo, es considerado muy positivo.

```
dea = 1.2 * Δ (MyDeaths_inicioComport, MyDeaths_finalComport).
kil = Δ (DeathsToWin_inicioComport, DeathsToWin_finalComport).
extra1 = + 2 sino mata.
extra2 = + 2 sino le matan.
hea = (Δ (Health_inicioComport, Health_finalComport))/100.
NuevaAdaptabilidad = (Antigua - dea + kil + hea + extra1 + extra2)/10.
```

4.2. Experimentos

4.2.1. Experimento 1

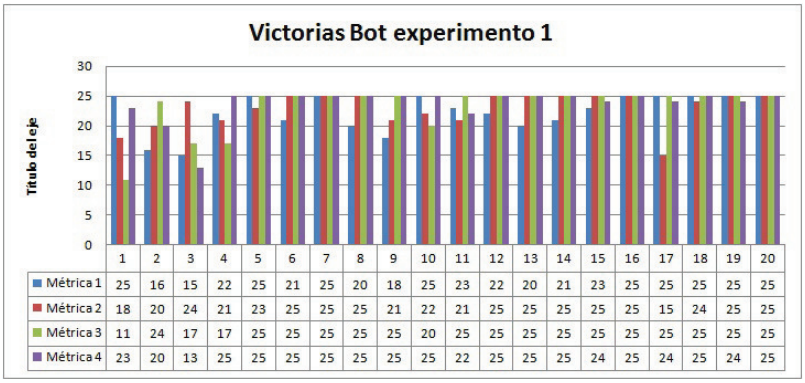
En el primer ejemplo se utilizan unas condiciones de partida para tener una referencia en futuros experimentos. A partir de ahí el resto de ejemplos llevarán al límite las condiciones de experimentación, la ventana (sección 4.1.1) y las métricas (secciones 4.1.2 y 4.1.3) buscando la mejor manera de aprovechar al máximo la técnica CBR sin perder de vista los objetivos de ganar partidas con el mayor éxito posible y de obtener el patrón óptimo de juego.

Condiciones

- Tamaño de ventana: 50 casos.
- Métricas para elegir caso: Métrica 1, 2, 3 y 4.
- Métrica premio/castigo: Matar siempre.

Discusión

Este primer experimento trata de mostrar el comportamiento del bot bajo unas condiciones teóricamente buenas, ya que el juego consiste en matar más que los demás. Como se puede ver en la Figura 4.1 el objetivo final del juego se consigue holgadamente, ya que en la mayoría de casos el bot gana la partida. A partir de la 5ª partida siempre hay alguna métrica que permite al bot ganar.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Métrica 1	25	16	15	22	25	21	25	20	18	25	23	22	20	21	23	25	25	25	25	25
Métrica 2	18	20	24	21	23	25	25	25	21	22	21	25	25	25	25	25	15	24	25	25
Métrica 3	11	24	17	17	25	25	25	25	25	20	25	25	25	25	25	25	25	25	25	25
Métrica 4	23	20	13	25	25	25	25	25	25	25	22	25	25	25	24	25	24	25	24	25

Figura 4.1: Victorias del bot

Tal como se observa en la Figura 4.2 los comportamientos predominantes en un bot ganador, como es este caso, son los de ataque, frente a los comportamientos defensivos que en ningún caso logran superar a los primeros. Esto demuestra que la naturaleza del juego obliga al bot a comportarse de manera ofensiva (o muy ofensiva) si lo que quiere es ganar. Es importante reseñar que dentro de los comportamientos de ataque, los más agresivos son mejor valorados que los demás (salvo en la métrica 1 que prefiere un comportamiento no tan agresivo), demostrando que para este tipo de condiciones el bot no se tiene que preocupar de esquivar ataques.

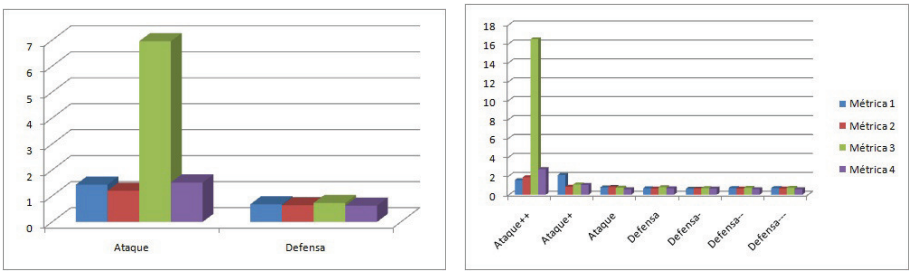


Figura 4.2: Comportamientos de ataque y defensa

Otra conclusión extraída de estos datos es la reflejada en la Figura 4.3. La primera de las gráficas muestra la igualdad existente entre las dos maneras de recoger objetos, salvo en el caso 3, donde (tal como muestra la Figura 4.4), hay un caso que sobresale. Esto demuestra que la tarea de recoger objetos no establece preferencias entre unas formas u otras y que depende, única y exclusivamente, del tipo de comportamiento. En la segunda de las gráficas de la Figura 4.3, se muestran las mejores formas de elegir arma. En este caso todavía no se puede establecer un indício de patrón ya que aunque en la métrica 3 sobresale uno de los elementos, no es menos cierto que esa variación es provocada por un único comportamiento. En el resto de casos las elecciones son bastante dispares.

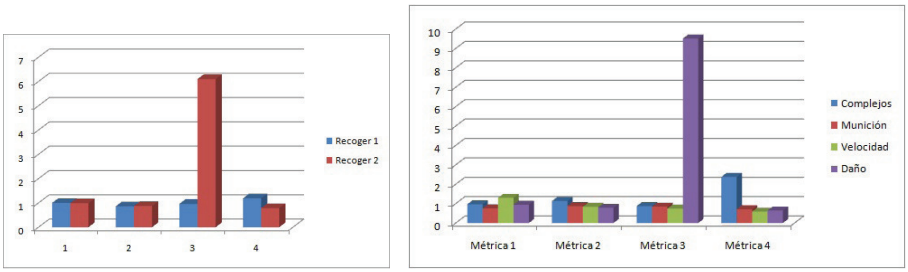


Figura 4.3: Recoger objetos y Tipos de ataque

Métrica 1		Métrica 2		Métrica 3		Métrica 4	
Adapt.	Comportam.	Adapt.	Comportam.	Adapt.	Comportam.	Adapt.	Comportam.
12,489076	Ataque+ 32	6,33987501	Ataque++ 12	213,193077	Ataque++ 42	24,988195	Ataque++ 11
5,04173987	Ataque+ 11	6,11278549	Ataque++ 31	6,04575293	Ataque++ 41	6,10541559	Ataque++ 12
4,07452472	Ataque++ 31	3,63949998	Ataque++ 22	1,778112	Ataque+ 22	2,21952	Ataque+ 12
4,01810325	Ataque++ 42	2,10607713	Ataque++ 21	1,7258966	Ataque+ 42	1,470144	Ataque+ 21
3,06537317	Ataque++ 41	2,01034774	Ataque+ 11	1,61244298	Ataque+ 11	1,3888	Defensa 12

Figura 4.4: Inicio base de casos

Los resultados pueden parecer óptimos, ya que se ganan bastantes partidas ($\frac{13}{20}$), pero el hecho de tener una ventana de 50 elementos sobre los 112 posibles hace que casos identificados como poco adecuados aparezcan en la ventana de trabajo. Esto provoca que aunque algunos de los resultados pudieran indicar hacia donde iría el patrón de juego, no se puede generalizar todavía, dado el uso excesivo de casos peor valorados. Si se redujera el número de elementos de la ventana a la mitad (de 50 a 25 casos) se podría corregir este error propiciando un mejor uso de los casos y una generalización del patrón de comportamiento.

4.2.2. Experimento 2

Este segundo experimento trata de corregir el posible error inicial de tener una ventana demasiado extensa que permite que casos poco adecuados empeoren el resultado, en vez de centrarse únicamente en los buenos.

Condiciones

- Tamaño de ventana: 25 casos.
- Métricas para elegir caso: Métrica 1, 2, 3 y 4.
- Métrica premio/castigo: Matar siempre.

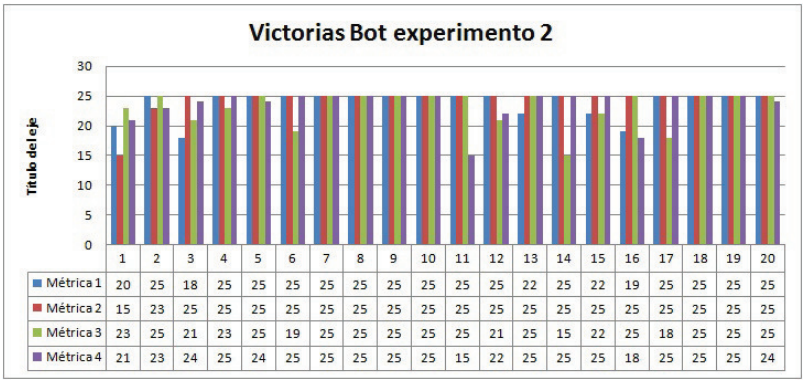
Discusión

Con estos cambios se logran establecer mejoras, ya que en todas las métricas probadas hay más variedad de casos con valores altos, tal como se ve en la Figura 4.5 o en el Anexo L. Esto demuestra que con una ventana de 25 casos, se discriminan los suficientes casos para que solo estén a disposición del bot los mejores casos. El primer número en los nombres de los comportamientos indica el tipo de arma elegida, (1) armas complejas, (2) armas con mucha munición, (3) armas rápidas y (4) armas dañinas y el segundo número indica la forma de recoger objetos, (1) salud/escudo y armamento y (2) salud, escudo y armamento.

Métrica 1		Métrica 2		Métrica 3		Métrica 4	
Adapt.	Comportam.	Adapt.	Comportam.	Adapt.	Comportam.	Adapt.	Comportam.
185,825829	Ataque++42	788,892871	Ataque++41	201,131006	Ataque++41	61,0113683	Ataque+42
1,97685412	Ataque++41	59,8228661	Ataque++42	2,0203132	Ataque++41	18,8574701	Ataque+32
1,85102386	Ataque++22	12,2210907	Ataque++31	1,70354459	Ataque42	2,34871849	Ataque++11
1,7136	Ataque22	2,57781885	Ataque++22	1,2544	Ataque+32	2,25827146	Ataque+12
1,51367313	Ataque+12	2,46733137	Ataque+42	1,21856	Ataque+41	1,70961458	Ataque++21

Figura 4.5: Inicio base de casos

Uno de los aspectos más importantes que hace que este experimento sea mejor que el anterior es comprobar que el objetivo final del juego (ganar partidas) sigue produciéndose con bastante regularidad ($\frac{15}{20}$) en todas las métricas (Figura 4.6), sin importar el nuevo tamaño de ventana. Con ese objetivo cumplido y con mayor uso de mejores casos se puede establecer que los resultados obtenidos serán más precisos que los anteriores y por lo tanto más representativos.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Métrica 1	20	25	18	25	25	25	25	25	25	25	25	25	22	25	22	19	25	25	25	25
Métrica 2	15	23	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25
Métrica 3	23	25	21	23	25	19	25	25	25	25	25	21	25	15	22	25	18	25	25	25
Métrica 4	21	23	24	25	24	25	25	25	25	25	15	22	25	25	25	18	25	25	25	24

Figura 4.6: Victorias del bot

La tendencia mostrada en este experimento (Figura 4.7) es que se extrema la diferencia entre los comportamientos de ataque y los defensivos, especialmente entre el comportamiento de ataque más agresivo y el resto, propiciado por una métrica "premio/castigo" (sección 4.1.3) muy radical. Por otro lado se puede observar qué tipo de arma es la mejor para ganar partidas. Tal como indica la segunda gráfica de la Figura 4.8, la elección de armas basada en el daño producido es la más exitosa, por delante incluso de las armas rápidas que inicialmente se podrían considerar mejores cuando lo que se quiere es disparar mucho y rápido. La primera gráfica de dicha figura, vuelve a resaltar que la forma de recoger objetos no tiene incidencia real en la elección, algo que también se puede ver en la Figura 4.5, que muestra como se intercalan los comportamientos que recogen los objetos de alguna de las dos maneras.

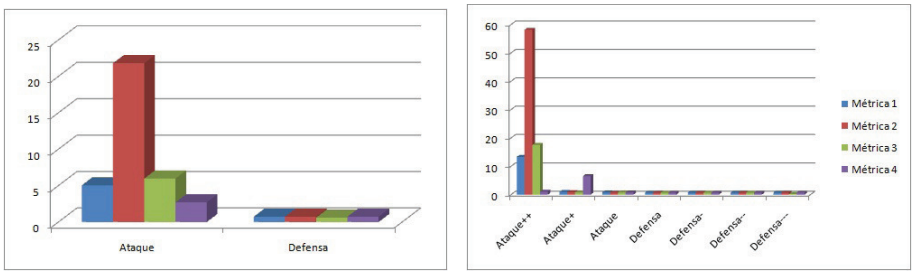


Figura 4.7: Comportamientos de ataque y defensa

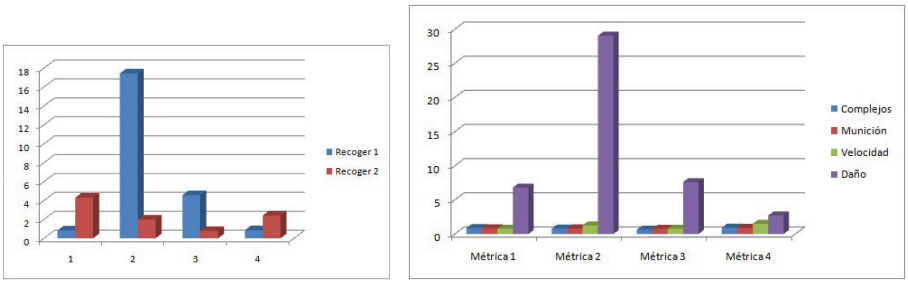


Figura 4.8: Recoger objetos y Tipos de ataque

Los cambios propuestos en este experimento han resultado satisfactorios ya que se ha mantenido el objetivo de ganar partidas y gracias a la reducción de la ventana a 25 casos se han podido establecer conclusiones sobre la mejor forma de actuar de manera más precisa, indicando lo que podría ser un posible patrón de juego. Aun así se puede analizar más críticamente el tamaño de ventana probando si con un número muy reducido de casos (5 casos) también proporciona resultados adecuados y útiles. Otra opción sería modificar la métrica "premio/castigo". Hasta ahora se premiaba "ir a matar" sin pensar nada más y aunque los resultados parecen óptimos, puede que un comportamiento algo más conservador siga dando los mismos o mejores resultados.

4.2.3. Experimento 3

Este tercer experimento se centra en llevar al extremo la reducción de la ventana.

Condiciones

- Tamaño de ventana: 5 casos.
- Métricas para elegir caso: Métrica 1 y 2.
- Métrica premio/castigo: Matar siempre.

Discusión

El resultado final es contradictorio. Muestra que si encuentra un entorno poco cambiante, puede ser muy provechoso (muchas victorias, Figura 4.9), pero no aprovecha al máximo las posibilidades de un sistema CBR.

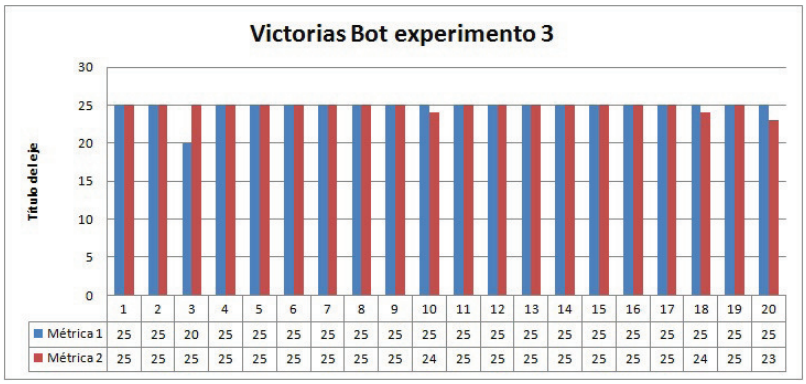


Figura 4.9: Victorias del bot

La Figura 4.10 enseña como realmente la situación no es muy diferente a las anteriores, beneficiando a los comportamientos de ataque sobre los otros.

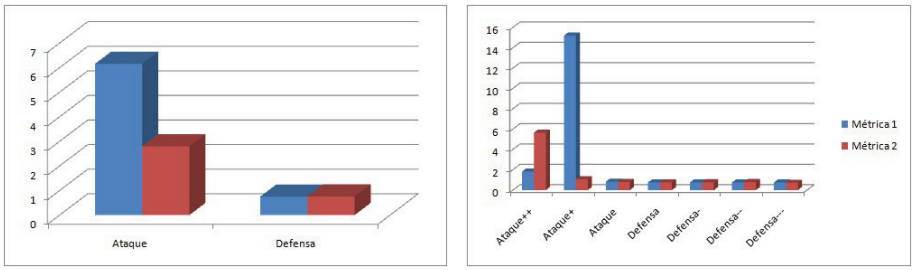


Figura 4.10: Comportamientos de ataque y defensa

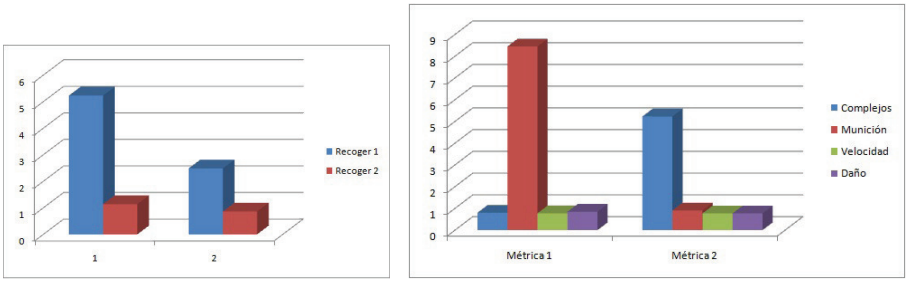


Figura 4.11: Recoger objetos y Tipos de ataque

El problema surge con la base de casos (Anexo L y Figura 4.12), ya que el número de casos que se han probado es muy reducido (para la métrica 2, solo hay dos casos que superan el valor medio). Si encuentra un comportamiento válido que ha sido útil muchas veces, el bot no dejará de usarlo aunque las situaciones del entorno sean distintas. Inicialmente no parece un problema (si tiene asignado un valor alto es que siempre o casi siempre ha sido útil), pero en el caso de que cambie la partida o el escenario y el comportamiento empiece a fallar, no tendrá margen de maniobra y seguirá usando el mismo caso hasta que mucho tiempo después haya visto reducido su valor y deje paso a otros comportamientos más adecuados. Una de las utilidades del sistema CBR es que permite tener una gran variedad de casos para según que situaciones, pero si la ventana es demasiado pequeña (5, en este ejemplo) esa variedad se ve limitada.

Métrica 1		Métrica 2	
Adapt.	Comportam.	Adapt.	Comportam.
143,238489	Ataque+ 21	54,1670419	Ataque++ 12
11,6170011	Ataque++ 22	3,57045168	Ataque+ 21
1,91594534	Ataque+ 42	1	Ataque+ 11
1,0732984	Ataque++ 31	1	Ataque+ 21
1,02	Ataque+ 32	1	Ataque+ 32

Figura 4.12: Inicio base de casos

4.2.4. Experimento 4

Con este experimento se intenta probar si premiando una actitud más conversadora, pero todavía ofensiva, se mejoran o igualan los resultados obtenidos en el experimento 2, hasta ahora el mejor de los obtenidos.

Condiciones

- Tamaño de ventana: 25 casos.
- Métricas para elegir caso: Métrica 1 y 2.
- Métrica premio/castigo: Evitar muerte, pero actuando.

Discusión

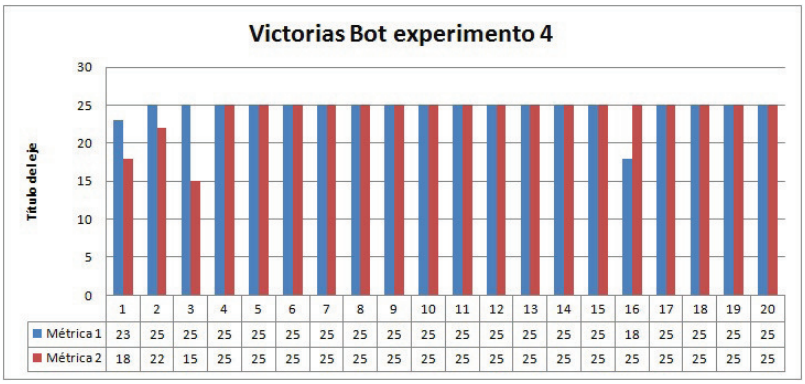


Figura 4.13: Victorias del bot

En este caso, el cambio a una métrica de "premio/castigo" que favorece menos que en los experimentos anteriores a los comportamientos de ataque, se siguen obteniendo similares resultados, dando a entender que si la métrica está destinada a ganar partidas, los mejores comportamientos son lo de ataque "muy agresivo" y la mejor selección de armas es la basada en el daño que producen. Especialmente significativo es que el comportamiento mejor valorado, con mucha diferencia sobre el resto, es el más agresivo. En un principio se podría llegar a pensar que eligiendo una métrica más conservadora, los comportamientos de ataque menos agresivos obtendrían mejores resultados, algo que por lo visto no es así. Viendo las Figuras 4.13, 4.14, 4.15 y 4.16, se observa que salvo pequeños detalles, los resultados obtenidos son similares a los del experimento 2.

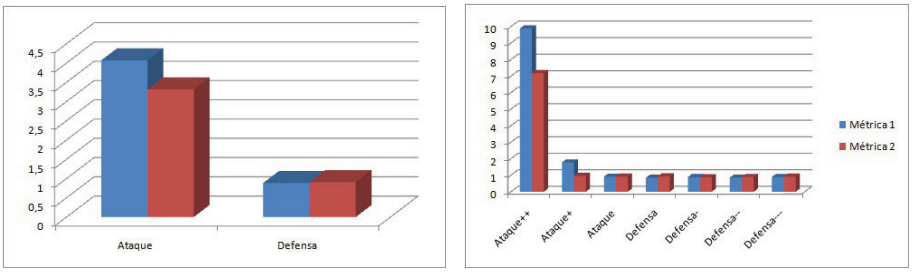


Figura 4.14: Comportamientos de ataque y defensa

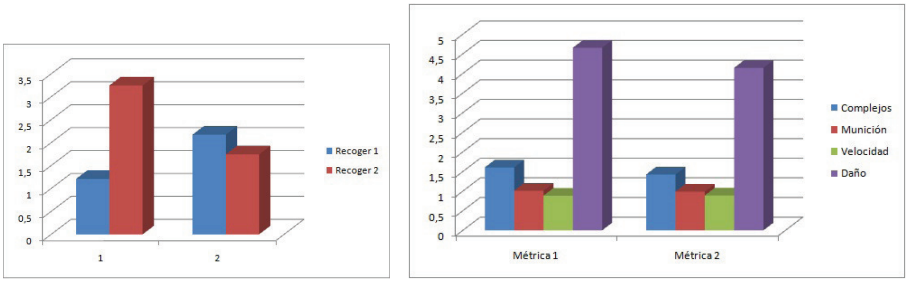


Figura 4.15: Recoger objetos y Tipos de ataque

Métrica 1		Métrica 2	
Adapt.	Comportam.	Adapt.	Comportam.
131,466967	Ataque++ 42	38,0450794	Ataque++ 41
11,2863867	Ataque+ 11	32,369309	Ataque++ 42
4,26827746	Ataque++ 11	20,8862697	Ataque++ 41
2,80774937	Ataque+ 21	8,42917624	Ataque++ 12
1,71072	Ataque++ 41	3,58367061	Ataque++ 22

Figura 4.16: Inicio base de casos

Las dos métricas "premio/castigo" probadas parece que son adecuadas pero sería interesante comprobar si con una métrica que variara mucho del objetivo final (matar enemigos) se consiguen igualmente resultados óptimos.

4.2.5. Experimento 5

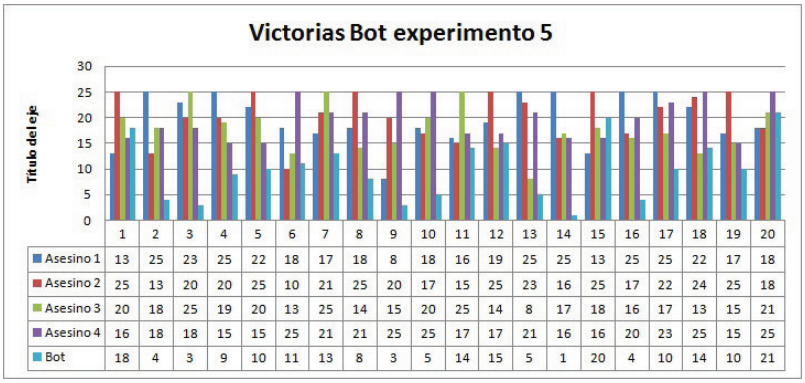
Este experimento intenta cambiar radicalmente la métrica "premio/castigo" (sección 4.1.3). Ahora premia que al bot no le maten, pero también que evite enfrentamientos, algo totalmente opuesto a la mecánica del juego. El resto de condiciones siguen siendo las consideradas óptimas.

Condiciones

- Tamaño de ventana: 25 casos.
- Métricas para elegir caso: Métrica 1.
- Métrica premio/castigo: Evitar muerte huyendo.

Discusión

Con este cambio radical de métrica "premio/castigo" lo primero que se puede observar rápidamente es que el objetivo de ganar las partidas no se cumple ($\frac{10}{25}$ muertes por partida y $\frac{0}{20}$ victorias). Esto es debido a que la nueva métrica va en contra de la esencia del juego. Si la clave de este juego es conseguir más muertes que el rival y se tiene una métrica que premia huir del enemigo y no atacar, es esperable que suceda lo que se ve en la Figura 4.17.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Asesino 1	13	25	23	25	22	18	17	18	8	18	16	19	25	25	13	25	25	22	17	18
Asesino 2	25	13	20	20	25	10	21	25	20	17	15	25	23	16	25	17	22	24	25	18
Asesino 3	20	18	25	19	20	13	25	14	15	20	25	14	8	17	18	16	17	13	15	21
Asesino 4	16	18	18	15	15	25	21	21	25	25	17	17	21	16	16	20	23	25	15	25
Bot	18	4	3	9	10	11	13	8	3	5	14	15	5	1	20	4	10	14	10	21

Figura 4.17: Victorias del bot

Otra de las conclusiones que arroja este experimento es el hecho de que los comportamientos defensivos no son útiles para este tipo de juegos. En la Figura 4.18 y Figura 4.19 se observa cómo los comportamientos defensivos son mayoritarios.

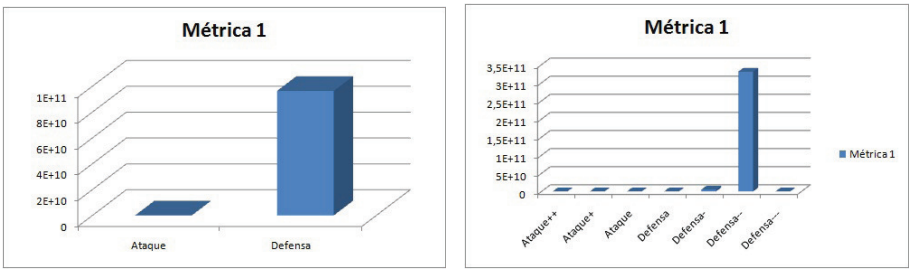


Figura 4.18: Comportamientos de ataque y defensa

Métrica 1	
Adapt.	Comportam.
1,32E+12	Defensa-- 31
2,5359E+10	Defensa- 42
377,58724	Ataque++ 22
3,69630806	Ataque 11
2,85326215	Ataque++ 21

Figura 4.19: Inicio base de casos

De la segunda gráfica de la Figura 4.20 se extrae que las armas más rápidas son las que más veces se usan en los comportamientos defensivos, pero como el objetivo final del juego (ganar) nose ha producido se puede concluir que no es la mejor forma de elegir armas aunque inicialmente se haya podido pensar lo contrario.

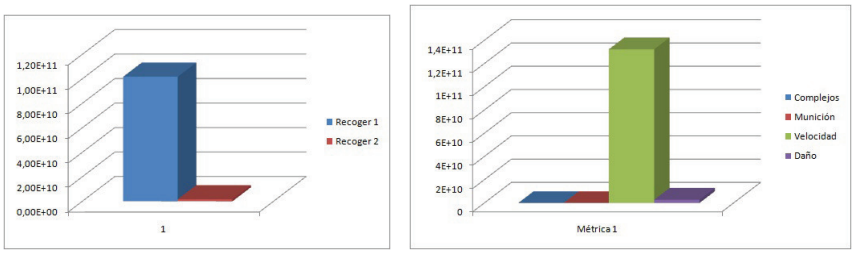


Figura 4.20: Recoger objetos y Tipos de ataque

4.3. Ideas generales

Después de esta extensa colección de experimentos se pueden extraer unos principios o ideas generales para los usuarios de este tipo de juegos, ya sea diseñando bots o jugando en persona.

1. El comportamiento que llevará a un bot o a un jugador a ganar partidas se debe basar en atacar de "manera muy agresiva". Se entiende como "manera muy agresiva", el hecho de atacar siempre que se pueda, aunque a la vez se sufran más ataques, ya que lo más importante, antes que evitar ataques, es estar en constante búsqueda de enemigos enfrentándose a ellos.

2. La mejor manera de atacar, como se ha podido constatar, es llevando el arma que más daño produzca. Esto iría en contra de una intuición previa que llevaría a pensar que un arma rápida es más útil en un entorno tan rápido y cambiante como es el del Unreal Tournament 2004.

3. La tercera idea clave es que la forma de recoger objetos en el juego no es significativo para su resultado. El bot ganador no debe preocuparse de recoger los objetos de salud y escudo por separado o a la vez. Solo de atacar y, cuando tiene problemas de salud o de arma, ir en busca del elemento más cercano.

4. Vistas las comparaciones entre métricas de selección de caso y los resultados obtenidos en las Figuras 4.4, 4.5, 4.12, 4.16 y 4.19 y su respectivas tablas extendidas del Anexo L, se puede establecer que no existen grandes diferencias entre unas métricas y otras. Es de destacar, sin embargo que, la métrica 1 de selección (que usa las 9 variables de entorno que controla el bot), al tener más variables para valorar los casos, puede elegir sin tener varios casos con el mismo valor. En esta situación, el valor de adaptabilidad determina cuál es el mejor.

Centrándose más en la técnica CBR, se puede extraer la conclusión de que la ventana de trabajo, que criba inicialmente los casos, debe ser de un tamaño medio (25 casos, aprox.) para evitar usar los peores casos (ventanas grandes, 50 casos, aprox.) y para evitar que haya poco margen para imprevistos si la situación del entorno cambia radicalmente (ventanas pequeñas, 5 casos, aprox.). Lo importante de esta técnica es que ofrece la posibilidad de tener una gran variedad de casos para diversas situaciones y que aquellos que han obtenido peores resultados son desterrados antes de hacer la elección, por lo que el tamaño de ventana medio ofrece un punto de equilibrio entre la gran variedad de casos y la criba de casos peor valorados.

Por último, viendo los resultados del experimento 5, se puede constatar que los algoritmos usados en este proyecto son adecuados. Inicialmente se podría pensar que las diferentes estrategias no tendrían gran reflejo en los resultados, que podrían ser muy parecidos, pero con este último experimento ha quedado demostrado que si las métricas elegidas no siguen la mecánica del juego los resultados no llevan al bot a su objetivo fundamental (ganar partidas).

Capítulo 5

Conclusiones y valoración personal

Este capítulo contiene las conclusiones extraídas durante el proceso de realización de este Proyecto Final de Carrera y unas propuestas de posibles líneas de trabajo futuro. Al final se presenta una valoración de lo que su realización ha supuesto a nivel personal.

5.1. Conclusiones

A lo largo de este proyecto se ha realizado un estudio acerca de si un sistema CBR basado en la hipótesis de los "marcadores somáticos" podría ser útil en el control de un bot. Este estudio se basaba en la idea de que el ser humano, en condiciones de toma de decisión, toma sus decisiones en función de unas experiencias pasadas y esas experiencias reducen las opciones disponibles antes de iniciar un proceso de deliberación.

Para ello, se ha seguido un proceso riguroso consistente en estudiar las herramientas disponibles, desarrollar el SomaticMarkerBot con sus comportamientos y establecer una serie de experimentos que probaran si los objetivos finales se habían conseguido. A continuación se va a comentar el proceso seguido en la realización del proyecto, resumiendo las partes más importantes y mostrando un resumen de las conclusiones que se han ido extrayendo, finalizando con un análisis final, con el que se pretende determinar si la hipótesis de trabajo planteada en 1.1 se ha conseguido.

Estudio de herramientas

Esta fase sirvió para familiarizarse con las herramientas y con la línea de investigación planteada. El trabajo desarrollado durante este periodo consumió unas 80 horas (16 % del trabajo global) de las dedicadas al proyecto, lo que demuestra la importancia que ha tenido en este caso la familiarización con el entorno.

Las dos herramientas usadas, aparte del juego, en este proyecto han sido la plataforma Pogamut y el gestor de bases de datos SQLite.

La plataforma Pogamut como se comenta en la sección 2.1, es un conjunto de librerías que permiten establecer la comunicación con el videojuego Unreal Tournament 2004, sin necesidad de aprender a manejar su lenguaje de scripting (UnrealScript[1]), ni su sistema de comunicación, ya que con estas librerías se pueden crear bots del juego en java.

[1] http://en.wikipedia.org/wiki/UnrealScript

Después de haber trabajado con esta plataforma se puede decir que aún es muy incompleta. Los autores de dichas librerías son los mismos que explican su funcionamiento en [17]. Durante el desarrollo del proyecto hemos pasado de la versión 2.0, muy simple, a la versión 3.1, bastante más completa, pero sin excesivos comentarios en el código que lo hiceran entendible. Aun así, una vez que se adaptó el sistema a nuestro proyecto, se pudo comprobar (sección 2.1.1) cómo los pasos para construir un bot son bastante básicos al tener únicamente cuatro clases principales: (1) UT2004BotRunner que se encargar de lanzar el thread del bot; (2) PogamutException que maneja todas las excepciones que puedan surgir en el código; (3) UT2004BotModuleController (sección 2.1.1) que contiene el controlador principal del juego; y (4) WorldView (sección 2.1.1) que controla los eventos que se producen en el entorno del juego.

Al crear el bot se empieza a entender, finalmente, cómo funciona el sistema, especialmente las clases UT2004BotModuleController y WorldView. Como se puede ver en los apartados UT2004BotModuleController y WorldView (completado en Anexo G) de la sección 2.1.1, dichas clases manejan todo el conjunto de movimientos, de ataques y de memoria del bot, facilitando su uso, ya que sin estas librerías de Pogamut, la comunicación del bot con el juego se realizaría por mensajes síncronos y asíncronos. Estos mensajes síncronos y asíncronos, en Pogamut están contenidos en clases individuales (Anexo F) que permiten a UT2004BotModuleController y WorldView tratar con ellos de una manera simple. Por eso se puede afirmar que el principal problema de trabajar con esta plataforma ha sido su constante cambio y su falta de documentación, pero que una vez comprendida, ha resultado más cómodo crear nuevos datos a partir de los existentes, siempre y cuando la comunicación entre Pogamut y el sistema fuera correcta, ya que más de una vez los datos pedidos al entorno eran devueltos incompletos o erróneos.

El gestor de bases de datos SQLite (sección 2.2) es un gestor muy simple y rápido en tiempos de ejecución. No funciona como una base de datos cliente-servidor, sino que el motor enlaza con el programa reduciendo su tiempo de latencia. Después de analizarlo se puede confirmar que la velocidad de la que hace gala consigue que las entradas/salidas de la base de datos no interfieran en la partida.

El último paso en este estudio de herramientas se dedicó a probar si se podrían usar algunas herramientas "pre-fabricadas" que facilitaran el trabajo, especialmente en la metodología del sistema CBR. En la sección 2.3 se muestran las herramientas consultadas. Se pudo concluir que aunque las ideas de alguna de ellas (jColibri2) eran bastante interesantes, introducir una nueva herramienta al sistema suponía ralentizar el proceso. En un entorno en "tiempo real" (0.25 segundos de ciclo de ejecución), como el de este juego, se necesitan las mínimas herramientas posibles, por este motivo se desechó especialmente el uso de jColibri2, el cual ofrecia una metodología completa de un sistema CBR.

Desarrollar SomaticMarkerBot

Después de seleccionar estas herramientas, se llevó a cabo el desarrollo del personaje (40 % del trabajo global), siguiendo los siguientes pasos: (1) se establecieron las condiciones en las que se iba a desarrollar la partida; (2) se determinaron qué variables de entorno iban a dar información al bot; y (3) se organizó la base de comportamientos del bot.

Durante la selección de condiciones de la partida (sección 3.1) se han regulado las variables que podían controlarse desde el servidor antes de lanzar una partida. Después de haber elegido aquellas más adecuadas se puede asegurar que el número de participantes elegido y el nivel de los rivales ha sido el adecuado puesto que han permitido evolucionar al bot, dejandole ganar una gran cantidad de partidas. Con un número distinto de bots o con diferentes niveles en los bot, estos no ganaban (por lo tanto no tenían casos valorados como positivos), o ganaban muy fácilmente (impidiendo un buen eso del sistema CBR).

Con las variables de entorno ha habido un gran problema de dimensionalidad. Un bot recibe más de 100 potenciales inputs de información del entorno (Anexo H) de las cuales la mayoria son datos superfluos o demasiado técnicos (velocidad de giro del bot en el aire, gravedad en el entorno actual...). Finalmente en la sección 3.2 se muestran las 9 variables de entorno elegidas, consideras como primordiales: muertes para ganar, si ve enemigos, tiempo para terminar, cantidad de escudo, salud y munición actual, muertes sufridas, número de armas en el inventario, cantidad de munición de dichas armas y controladores de estado (si ha muerto recientemente o ve un proyectil cerca). Estas variables permiten tener un control simple pero completo de la escena y el bot conoce todos sus signos vitales (salud, armas...) y lo que más le interesa del entorno (ver enemigos, ver proyectiles...). Uno de los mayores problemas ha sido seleccionar cuáles de las más de 100 informaciones recibidas se debían utilizar (hasta solo 9).

Los diferentes comportamientos del bot supusieron la parte fundamental del desarrollo. Inicialmente se pensó que los comportamientos deberían ser dinámicos, pero una vez entendido el sistema y la información que facilitaba el juego, se vio claramente que no era posible, por lo que una parte fundamental del sistema CBR se tuvo que modificar. Por este motivo se entendió que había que crear los comportamientos desde el inicio y lo más variados posible para que dieran al bot el máximo dinamismo posible. En la sección 3.3 se muesta cómo funciona el sistema CBR en el SomaticMarkerBot, y como la hipótesis de los "marcadores somáticos" se ve reflejada. El proceso es el mismo que un sistema CBR típico (sección 1.3.1) salvo que los casos no se modifican ni retocan en ningún momento ya que los comportamientos son pre-establecidos. Lo único que hace variar la posición de un caso en la memoria del bot son sus experiencias pasadas, y tal como se explica en la sección 3.3, el valor de adaptabilidad es el encargado de evaluar esas experiencias. Dicho valor de adaptabilidad sería el "marcador somático" que determina lo bueno o malo que es un caso.

Con los comportamientos se tomó la decisión de que fueran árboles binarios y que a partir de las informaciones que se reciben del entorno, se fueran desplazando hacia las hojas, donde se encuentran las "acciones simples". Como se explica en la sección 3.3.1 las "acciones simples" son las que determinan que va a hacer el bot a más bajo nivel (andar, huir, atacar, recoger objetos). Estas acciones simples también son muy extensas, ya que no hay una única forma de atacar o de huir. El conjunto total de posibles combinaciones (Anexo J) se redujo a un número mucho menor: 21 (Tabla 3.4). Con esta simplificación se han generado los comportamientos. Éstos también han tenido que reducirse, porque con el número de niveles y de hojas de un árbol variable, se podian llegar a generar $\sim 5 * 10^{19}$ comportamientos distintos y eso sin tener en cuenta que el orden de las ramas y hojas del árbol también generaría algunos más.

Finalmente se llego a la decisión de crear 16 tipos de árboles de comportamiento (Anexo K), los cuales se podrían diferenciar por: 2 formas diferentes de recoger objetos (salud, escudo y armamento, o salud/estudo y armamento), 4 formas de ataque (para armas rápidas, armas con mucho daño, armas con mucha munición y una mezcla de ataques arriesgados, como, por ejemplo, atacar al más lejano o al que más muertes lleva asignadas). Además, para tener mayor variedad de comportamientos se establecieron dos grupos diferentes, de ataque y defensivos. Dentro de estos tipos de comportamientos hay divisiones internas: Para los comportamientos de ataque existen tres variedades que van desde los poco agresivos (atacan con velocidad normal y sin correr), hasta los muy agresivos (atacan con velocidad máxima y corriendo siempre) y para los comportamientos defensivos se describieron cuatro variedades que iban desde los poco precavidos (tiene la misma preferencia entre huir y atacar), hasta los muy precavidos o huidizos (intenta huir siempre de sus enemigos, salvo que la situación sea muy favorable).

En conclusión, el desarrollo de los comportamientos nos planteó el problema de que debían ser determinados antes de empezar y que había muchas variables que reducir hasta tener un conjunto de datos manejable. La ventaja es que si se aprovecha las experiencia de jugadores expertos al crearlos, se elimina aquella fase de un sistema CBR donde casi siempre tiene que participar un ser humano, la revisión.

Experimentos

Una vez el bot estaba construido, se han diseñado experimentos que demostraran si los objetivos iniciales se cumplían. Estos experimentos tenían tres condiciones que variaban de uno a otro buscando diversas condiciones para las pruebas: (1) ventana de trabajo (sección 4.1.1) que contiene los primeros N casos ordenados por el valor de adaptabilidad; (2) métricas de selección de caso (sección 4.1.2) que se encargan de elegir un único caso entre los situados en la ventana; y (3) métricas de "premio/castigo" (sección 4.1.3) que se encargan de valorar si un comportamiento ha sido bueno/malo modificando su valor de adaptabilidad. Cada experimento tiene su propia ventana de trabajo y su propia métrica "premio/castigo" y dentro de cada experimento varía la métrica de selección de caso para establecer las comparaciones entre ellas.

Tras la evaluación de dichos experimentos se pueden extraer un conjunto de conclusiones, tal como se señalan a continuación:

- Según lo visto en el experimento 1 (sección 4.2.1: ventana de 50 casos y métrica premio/castigo centrada en "matar siempre") , el uso de una ventana de 50 casos sobre los 112 posibles provoca que casos valorados como negativos no desaparezcan de la ventana de trabajo por lo que se siguen usando y ralentizan al bot en su objetivo de conseguir victorias (aun así consigue una media de $\frac{13}{20}$ victorias), y casos valorados como positivos no se usan tanto como deberían. La métrica de "matar siempre" provoca que el mejor comportamiento sea el de ataque agresivo. También se observa, desde este primer experimento, que las dos opciones para recoger objetos resultan indiferentes para el bot.

- El experimento 2 (sección 4.2.2: ventana de 25 casos y métrica premio/castigo centrada en "matar siempre") demuestra que una ventana de trabajo más reducida que la anterior consigue que los casos peor valorados desaparezcan de la ventana, de este modo los resultados son mejores ($\frac{15}{20}$ victorias) y más fiables porque en el momento de elegir un caso los únicos que se encuentran en la ventana son aquellos mejor valorados. Con este experimento se observa un indicio bastante fiable de que los comportamientos de ataque muy agresivos y las armas que provocan más daño son las mejores opciones de juego.

- El experimento 3 (sección 4.2.3: ventana de 5 casos y métrica premio/castigo centrada en "matar siempre") incide en el hecho de seguir reduciendo la ventana de trabajo. Los resultados son engañosos ya que muestran que el resultado final este experimento es el mejor de todos ($\frac{19}{20}$ victorias), pero con una ventana de 5 casos el bot está desprotegido ante un cambio radical de situaciones o escenario, puesto que los mejores casos tardarían mucho en reducir su valor para dar entrada a nuevos casos en la ventana.

- El cuarto experimento (sección 4.2.4: ventana de 25 casos y métrica premio/castigo centrada en "evitar muerte, pero actuando") demuestra que un forma de premiar al bot menos agresiva que las anteriores ("matar siempre") sigue funcionando igual o mejor ($\frac{16}{20}$ victorias) que en el experimento 2. Incluso los comportamientos se mantienen igual de agresivos aunque no se les premie tanto, como en los experimentos anteriores. La forma de recoger objetos sigue dependiendo de otros factores y la mejor forma de atacar es con las armas que más daño causan, evitando en todo momento el resto de combinaciones, incluida la de armas rápidas que inicialmente eran consideradas la mejor opción.

- El último experimento (sección 4.2.5: ventana de 25 casos y métrica premio/castigo centrada en "evitar muerte huyendo") demuesta que, si en los casos anteriores con métricas de ataque ("matar siempre" y "evitar muerte, pero actuando") el bot ganaba partidas, con una métrica opuesta, priorizando lo defensivo, el bot es incapaz de ganar ($\frac{0}{20}$ victorias, $\frac{10}{25}$ muertes por partida), debido a la métrica opuesta a la esencia del juego (conseguir más muertes que los rivales).

Hipótesis de trabajo: ¿conseguida?

Como resumen de esos experimentos se obtienen estas conclusiones que determinan que los objetivos iniciales se cumplen, tal como se señala a continuación:

- El sistema CBR y la hipótesis de los "marcadores somáticos" es muy útil en el control de bots en entornos cambiantes. La base de casos se va acomodando a las situaciones que surgen y permiten que el bot con las condiciones de partida establecidas inicialmente, acabe ganando las partidas sea cual sea la situación presenciada. Esta conclusión muestra la importancia que pueden tener este tipo de sistemas en la creación de bot más humanos. Superando algunas limitaciones (modificación de comportamientos en tiempo de ejecución y movimientos menos mecánicos) este sistema podría ser el futuro en la creación de bots, dada su fácil adaptación a entornos cambiantes.

- También se han obtenido conclusiones claras sobre cuáles son las mejores condiciones de un bot para ganar partidas. Los bots que van al ataque con el arma que más daño cause tienen un mayor porcentaje de probabilidades de ganar las partidas (77,5 % de victorias entre el experimento 2 y 4), siempre y cuando las métricas elegidas para premiar/castigar al bot sea coherente con la mecánica del juego (conseguir más muertes que los rivales). La forma de atacar siempre debe ser la más agresiva de todas (corriendo y a máxima velocidad), incluso aunque la métrica "premio/castigo" favorezca más las actitudes menos agresivas. Las armas que más daño causan en todo momento se consideran como la mejor opción, aunque inicialmente dada la mecánica del juego se pudiera pensar que las armas más rápidas pudieran ser mejores. La forma de recoger objetos resulta indiferente, ya que el bot se preocupa, única y exclusivamente, de atacar. Las métricas de selección de caso no establecen grandes diferencias por las que decantarse entre unas y otras, lo importante es que cuando dos casos tengan la misma importancia en una situación se elija el que las experiencias pasadas tenga considerado como mejor (valor de adatabilidad más alto).

5.2. Trabajo o líneas futuras

Tras la realización de este proyecto se proponen a continuación algunas de las posibles líneas de trabajo futuras:

- Tener en cuenta más variables de entorno, especialmente aquellas consideradas de nivel de avanzado ya que podrían mejorar los resultados.

  - Conocer la invulnerabilidad de los rivales (no es recomendable atacar a un rival que sea invulnerable). Si el bot es invulnerable puede atacar con más frecuencia de lo estándar.
  - Determinar en qué tipo de suelo se encuentra, no es lo mismo estar en el agua que en tierra firme.
  - Conocer la velocidad de movimientos, la gravedad y cualquier otro elemento físico del entorno que ayude a comprender mejor por donde se mueve el bot.

- Añadir alguna técnica de IA, especialmente a las acciones simples, para que los movimientos del bot sean más humanos.

- Trabajar con las nuevas actualizaciones de Pogamut, las cuales permiten obtener más y mejores datos, de esta manera se podría depurar el código obteniendo seguramente mejores resultados, especialmente en lo referente a movimientos y disparos del bot.

- Conseguir la modificación en tiempo real de los comportamientos, en vez de tenerlos definidos e inalterables (salvo la adaptabilidad) desde el inicio. De esta manera se podrían crear nuevos comportamientos o modificar los existentes y tener una base de casos más evolucionada en el tiempo.
- Introducir nuevas hipótesis interesantes derivadas de las ya existentes. Como es el caso de la "serendipia", malas decisiones que traen buenos resultados [19].

5.3. Valoración personal

El balance general al realizar este proyecto ha sido positivo.

La posibilidad de realizar un proyecto completo de principio a fin me ha permitido tener una visión global de las partes que éste entraña. Se han podido poner en prácticas bastantes cosas aprendidas durante la carrera, sobre todo en el aspecto de desarrollo de proyectos.

La posibilidad de descubrir nuevas técnicas de inteligencia artificial, que hoy en día están muy presentes, ha servido para comprender que aunque estamos muy lejos de conseguir robots de apariencia "humana" se avanza en una línea muy prometedora, ya que como se puede ver en las conclusiones del proyecto, la aplicación de estas técnicas sirven para adaptar el bot a cualquier entorno, algo considerado genuinamente humano.

Respecto a las dificultades encontradas en el desarrollo de este proyecto considero que ha habido dos dificultades principalmente: (1) comprender herramientas hechas por otras personas sin una documentación adecuada y en constante evolución y, (2) reducir el entorno a un espacio de trabajo limitado sin perder la globalidad del entorno.

Glosario

Juegos de aventuras: En este género, el usuario se mete en la piel de un personaje que debe avanzar a través de una o varias tramas que conforman la historia del juego.

Juegos de deportes: Abarcan una gran cantidad de deportes (reales y ficticios) en los que el usuario compite por ser el mejor en diversas competiciones.

Juegos educativos: Tienen como objetivo prioritario potenciar algún aspecto pedagógico. Suelen ir orientados al público más joven.

Juegos de estrategia: Son aquellos en los que el usuario se encarga de gestionar un gran número de recursos de forma simultánea para la consecución de los objetivos. La acción de todos los usuarios puede realizarse en tiempo real (todos al mismo tiempo, RTS) o por turnos (cada usuario tiene un marco de tiempo para realizar una acción mientras los demás esperan su turno). Suelen ser juegos de gestión (control de parques, hospitales, etc) o bélicos.

Juegos de shooter: Juegos de carácter bélico en los que el objetivo del usuario es abrirse camino a través del juego disparando a todo jugador que se ponga a tiro.

Juegos de lucha: Género en el que los combates cuerpo a cuerpo cobran especial protagonismo.

Juegos de "Mesa": Esta categoría está compuesta por todos aquellos videojuegos encargados de emular los tradicionalmente conocidos como "juegos de mesa", juegos físicos que cuentan con un tablero y fichas (ajedrez, mahjong, etc).

Juegos de "Party": Este tipo de juegos se centra en el entretenimiento en grupo. Normalmente se utiliza un modo de juego por turnos, en el que gana el usuario que mayor puntuación consiga al cabo de un número determinado de etapas.

Juegos de plataformas: Es una de las categorías más antiguas. El usuario encarna a un jugador que avanza por un mundo en el que debe saltar, correr y alcanzar determinados elementos repartidos por el entorno si quiere alcanzar su objetivo. Los hay tanto bidimensionales como tridimensionales, pero los juegos de culto de este género son del tipo 2D.

Juegos de rol: En ellos, el usuario representa un papel y debe interactuar con el entorno y el resto de jugadores del videojuego para poder mejorar sus características.

Juegos de simulación: Este tipo de juegos se encargan de reproducir una escena cotidiana (cuidar una mascota, conducir un determinado tipo de vehículo, dirigir la vida de un personaje, etc.).

NPC: Siglas del término inglés "Non-Player Character". Utilizado como sinónimo de bot y agente autónomo.

RTS: Siglas del término inglés "Real-time Strategy". Son videojuegos de estrategia en los que no hay turnos sino que el tiempo transcurre de forma continua para los jugadores.

TCP/IP: Modelo de descripción de protocolos de red. El modelo TCP/IP, describe un conjunto de guías generales de diseño e implementación de protocolos de red específicos para permitir que una computadora pueda comunicarse en una red. TCP/IP provee conectividad de extremo a extremo especificando como los datos deberían ser formateados, direccionados, transmitidos, enrutados y recibidos por el destinatario.

UnrealScript: Lenguaje pensado exclusivamente para desarrollar contenido para juegos que usen el motor Unreal. Está basado en Java y C++. Es un lenguaje de programación orientado a objetos (OOP - Object Oriented Programming) lo que significa que está basado en los conceptos de clases y herencias. Es importante hacer notar que UnrealScript no tiene todas las características de un lenguaje de programación más completo como pueda ser Java; principalmente no soporta conceptos como herencias múltiples.

Thread de ejecución: Característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos thread de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente. Un thread es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea.

Protocolo de comunicación: Conjunto de reglas usadas por computadoras para comunicarse unas con otras a través de una red.

Cliente/Servidor: Modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, que le da respuesta.

Latencia: Suma de retardos temporales dentro de una red. Un retardo es producido por la demora en la propagación y transmisión de paquetes dentro de la red.

Framework: Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un problema particular, que sirve como referencia para enfrentar y resolver nuevos problemas similares.

Mutadores: Modificaciones especiales que permiten al juego cambiar su forma, incluso saliéndose de lo considerado común.

Bibliografía

[1] Turing, A.M. Computing Machinery and Intelligence. Mind 49: 433-460. 1950. 1.3

[2] Russell, Stuart J. and Norvig, Peter. Artificial Intelligence: A Modern Approach (2nd ed.). Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2. 2003.

[3] Hilera, José R. and Martínez, Víctor J. "Redes neuronales artificiales". Alfaomega. Madrid. España. 2000.

[4] Kolodner, Janet L. Case-based Reasoning. San Francisco, CA: Morgan Kaufmann. ISBN 1-55860-237-2. 1993. 1.3.1

[5] Riesbeck, C. K. and Schank, R. C. Inside Case-Based Reasoning. Lawrence Erlbaum Associates, Cambridge, MA. 1989. 1.3.1

[6] Aamodt, Agnar, and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. In Artificial Intelligence Communications 7, no. 1 (1994): 39-52. 1994. 1.3.1, D.1.1

[7] Ontañon, S. and Mishra, K. and Sugandh, N. and Ram, A. Case-Based Planning and Execution for Real-Time Strategy Games. In International Conference on Case-based Reasoning (ICCBR 2007). 2007. 1.3.1

[8] Weber, B.G. and Mateas, M. Case-Based Reasoning for Build Order in Real-Time Strategy Games. In Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2009). 2009. 1.3.1

[9] Weber, B.G. and Mateas, M. Conceptual Neighborhoods for Retrieval in Case-Based Reasoning. In Proceedings of the International Conference on Case-Based Reasoning (ICCBR 2009), 343-357. 2009. 1.3.1

[10] Sánchez-Pelegrín, R. and Gómez-Martín, M.A. and Díaz-Agudo, B. A CBR Module for a Strategy Videogame. 1st Workshop on Computer Gaming and Simulation Environments, at 6th International Conference on Case-Based Reasoning (ICCBR 2005). 2005. 1.3.1

[11] Floyd, M.W. and Esfandiari, B. and Lam, K. A Case-based Reasoning Approach to Imitating RoboCup Players. In: Proceedings of FLAIRS-2008, Florida AI Research Symposium. 2008. 1.3.1

[12] Menkovski, V. An Artificial Intelligence aware model of turn based games and an AI implementation of a computer player for the Monopoly game. Tesis de Master, Abril 2008. 1.3.1

[13] Damasio, A.R. El error de Descartes: La emoción, la razón y el cerebro humano. Critica S.L., Barcelona. ISBN 978-84-8432-787-5. 2006. (document), 1.3.1, 8, 9

[14] Morris, De Groot. Optimal Statistical Decisions. Wiley Classics Library. 2004. (Originally published 1970.) ISBN 0-471-68029-X. 2004. 10

[15] Hoogendoorn, M. and Merk, R. and Treur, J. A Decision Making Model Based on Damasio's Somatic Marker Hypothesis. 9th International Conference on Cognitive Modeling, ICCM'09. 2009. 1.3.1

[16] Pimentel, C.F. and Cravo, M.R. "Don't think too much!" – Artificial Somatic Markers for Action Selection. In: Affective Computing and Intelligent Interaction and Workshops, 2009. 2009. 1.3.1

[17] Gemrot, J. and Kadlec, R. and Bída, M. and Burkert, O. and Píbil, R. and Havlícek, J. and Zemcák, L. and Simlovic, J. and Vansa, R. and Stolba, M. and Plch, T. and Brom, C. Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. Agents for Games and Simulations: Lecture Notes in Computer Science, 2009, Volume 5920/2009, 1-15, DOI: 10.1007/978-3-642-11198-3_1. 2009. 2.1, 5.1

[18] Gebhard, Patrick. ALMA - A Layered Model of Affect. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05), 29-36, Utrecht, 2005. 2.1

[19] Aguilera Lizarraga, Miguel. Efectos no lineales de la integración de capacidades emocionales en agentes inteligentes. Proyecto de fin de carrera. Zaragoza, 2010. 5.2